NAVIGATION AMONG MOVABLE OBSTACLES IN UNKNOWN ENVIRONMENTS

A Thesis
Presented to
The Academic Faculty

by

Martin Levihn

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Computer Science

Georgia Institute of Technology
May 2011

NAVIGATION AMONG MOVABLE OBSTACLES IN UNKNOWN ENVIRONMENTS

Approved by:

Professor Mike Stilman, Committee Chair
School of Computer Science
Georgia Institute of Technology

Professor Mike Stilman, Advisor
School of Computer Science
Georgia Institute of Technology

Professor Henrik Christensen
School of Interactive Computing
Georgia Institute of Technology

Professor Irfan Essa
School of Interactive Computing
Georgia Institute of Technology

Date Approved: 30 March 2011

ACKNOWLEDGEMENTS

I want to especially thank Professor Mike Stilman for his in-depth thoughts on Navigation Among Movable Obstacles. I would further like to thank Neil Dantam for proofreading this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

This work presents a new class of algorithms that extend the domain of Navigation Among Movable Obstacles (NAMO) to unknown environments. Efficient real-time algorithms for solving NAMO problems even when no initial environment information is available to the robot are presented and validated. The algorithms yield optimal solutions and are evaluated for real-time performance on a series of simulated domains with more than 70 obstacles. In contrast to previous NAMO algorithms that required a pre-specified environment model, this work considers the realistic domain where the robot is limited by its sensor range. It must navigate to a goal position in an environment of static and movable objects. The robot can move objects if the goal cannot be reached or if moving the object significantly shortens the path. The robot gains information about the world by bringing distant objects into its sensor range.

The first practical planner for this exponentially complex domain is presented. The planner reduces the search-space through a collection of techniques, such as upper bound calculations and the maintenance of sorted lists with underestimates. Further, the algorithm is only considering manipulation actions if these actions are creating a new opening in the environment. In the addition to the evaluation of the planner itself is each of this techniques also validated independently.

CHAPTER I

INTRODUCTION

Robots would be much more useful if they could move obstacles out of the way.

In the near future robots will move out of the laboratories and into homes. Humans will then expect the robots to react to high-level directions such as "please open the door" or " please come to the kitchen". Given the command, it is expected that the task will be performed and only further information be required if something critical occurs, preventing a successful execution. The domain of Navigation Among Movable Obstacles moves the boundary of what is declared as preventing a successful execution by the robot closer to the one of humans. For example if a human wants to move from the living room to the kitchen but a chair is in the only doorway, the human would move the chair out of his way instead of declaring that it is not possible to reach the kitchen. Nevertheless would most robots nowadays simply declare failure in this case. However, even if another way into the kitchen would exist, humans might rather move the chair than take a long detour, again in contrast to robots.

The human decision of when to rather move the chair in the previous example rather than taking a detour is guided by the effort involved in either option. This notion can be described by a cost function, having a cost for both manipulation and pure navigation. Based on this cost function, the optimal, or lowest effort, option should be chosen.

This work therefore explores the problem of optimal Navigation Among Movable Obstacles (NAMO). NAMO is an important problem in motion planning because it gives mobile robots better ability to reason about the environment and choose to manipulate obstacles [23]. Robots that solve NAMO will accomplish tasks that are

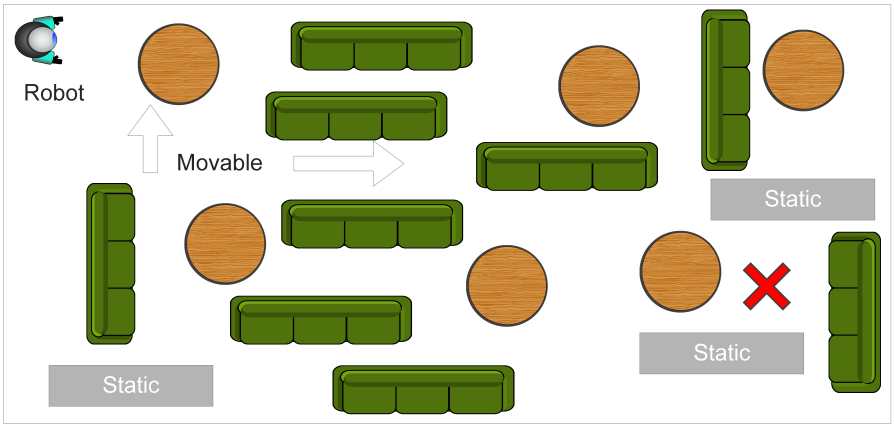
otherwise difficult or impossible. They will operate in cluttered human environments and strive towards human-level navigation. In order to accomplish this goal, motion planning must overcome a number of theoretical and practical challenges.

In contrast to most prior work in this field is this work exploring NAMO in practical scenarios where the robot attempts to reach a fixed goal position in a *reconfigurable and unknown environment*. Starting with no knowledge about the environment, the robot uses limited sensor information to locally detect objects and incrementally build a world model while simultaneously manipulating its environment. The robot may move objects if the goal cannot be reached or if moving the object would significantly shorten the path to the goal. The robot is required to always take the optimal action based on his current world knowledge as defined by a cost function.

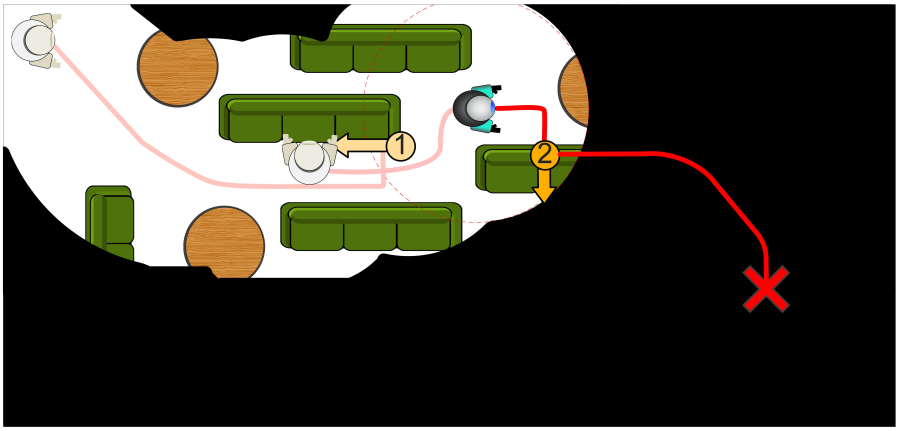
An illustrative example of this domain is given in Fig. 1(a) where the robot must alter the environment in order to navigate towards an otherwise unreachable goal. With only local and incomplete information, including the existence and movability of objects, the robot makes an optimal decision at each step based on acquired knowledge. As shown in Fig. 1(b), the robot gradually improves the world model as it navigates towards the goal, changing optimality conditions at each step.

1.1 Concepts

An illustrative example of this domain is given in Fig. 1(a) where the robot must alter the environment to navigate towards an otherwise unreachable goal. With only local and incomplete information, including the existence and movability of objects, the robot makes an optimal decision at each step based on acquired knowledge. As shown in Fig. 1(b), the robot gradually improves the world model as it navigates towards the goal. Every world model update can however affect the robots current plan. The current plan can become invalid through intersections with the new world information or the plans optimality can be affected.



(a) Map configuration: showing start, goal and obstacles.



(b) During execution: showing the first two manipulation actions.

Figure 1: Simulated demonstration of the NAMO algorithm for problems that require both navigation and manipulation without prior knowledge of the environment. Visible information in (b) is gathered online.

In order to find an optimal plan, all possible actions may have to be reevaluated whenever new information is perceived. Yet, iteratively recomputing the cost of all possible actions is infeasible for realistic domains since the computational complexity of planning is exponential in the number of known objects [23]. In practice, this naïve approach yields a runtime of 5 weeks, 2 days and 6 hours for a problem very similar to the one visualized in Fig. 1(a). This work presents a computationally feasible strategy that accounts for environmental changes, reducing the runtime for the same domain to seconds, while still ensuring optimal decision making.

1.1.1 Primary Concepts

This work introduces three major concepts that reduce the search space for NAMO in Unknown Environments while perserving optimality.

1.1.1.1 Replanning

The proposed algorithm identifies cases where new information does not affect previous calculations. Instead of reevaluating all actions when new obstacles are detected, the algorithm only performs additional computation when information affects with the optimality of the existing plan.

1.1.1.2 Obstacle Evaluation

An upper bound is calculated and maintained that limits the number of manipulation actions that have to be evaluated for each obstacle. The upper bound is given by the global minimum cost plan at each time. The bound is checked against cost estimates of plans. If the estimate is exceeding the upper bound, no plans that have at least equal manipulation actions on the object are considered anymore.

Further are only plans considered that include manipulation actions were the manipulation actions create a new opening in the environment as well as one plan having no manipulation actions.

1.1.1.3 Obstacle Ordering

Not every object known to the robot is necessarily evaluated for possible manipulation actions. Rather the algorithm maintains two lists with underestimates and iterates over the lists until further iteration can not yield a lower cost plan. The lists associate obstacles with the minimum cost of all plans manipulating that obstacle. While both lists represent underestimates, the first list takes the world as currently known to the robot into account while the second list assumes free space. The list taking the world into account usually provides a tighter bound. However, the list has to be invalidated

if a manipulation action is performed after the list was created. This is because free space is created that was not assumed during the creation of the list. The second list, on the other hand is never invalidated since free space was already assumed at the time of the creation. This list can therefore be used for lazily recovering of the first list.

1.1.2 Secondary Concepts

This work further details on the navigation planning for NAMO in Unknown Environment. A slight modification of the D* Lite [10] algorithm is presented that allows its usage in the domain of Navigation Among Movable Obstacles in Unknown Environments. The D* Lite search-tree is updated if environment changes are detected, but an actual path is not always calculated by D* Lite since the navigation goals can change drastically in the NAMO domain. This is because the robot might have to navigate to a manipulation configuration rather than the goal configuration.

1.2 Outline

This work is structured in the following way. First, related work is discussed in the following chapter. Chapter 3 is then providing a formal problem formulation and state all the assumptions and restrictions made to the domain. Chapter 4 is introducing two algorithms that are both capable of finding the optimal solution, the *baseline* and *optimized* algorithm. Chapter 5 is providing the optimality proofs for the algorithm. For clarity, these two chapters omit secondary aspects, the detection of openings in the environment and the navigation planning. Chapter 6 and 7 discus these secondary aspects: opening detection and navigation planning, respectively.

Chapter 8 is providing detailed experimental results. The *baseline* and *optimized* algorithms are compared. Further, each of the optimization steps used in the *optimized* algorithm are evaluated independently.

Challenges of the domain itself as well as algorithm limitations are discussed in

chapter 9. The thesis concluds with future work in chapter 10 and final remarks in chapter 11.

CHAPTER II

PREVIOUS WORK

This chapter provides an overview of related work. First, work directly connected to the domain of Navigation Among Movable Obstacles is discussed. The second section of this chapter focuses on general motion planning in unknown environments.

2.1 Navigation Among Movable Obstacles

Problems involving movable obstacles with *complete environment information* pose a significant computational challenge. Wilfong [29] first proved that motion planning among movable obstacles is NP-hard. Demaine [3] further proved that even the simplified version of this problem, in which only unit square obstacles are considered, is also NP-hard. Chen [2] designed the first planner that handled multiple movable objects and a navigation goal. The heuristic planner first generated a series of subgoals and solved the subgoals separately by a local planner. However, Chen's planner did not address problems where the order of object manipulations decides the solution.

Okada [17] presented a planner for a humanoid robot that is capable of finding a navigation path from a given goal location to a start location while manipulating movable obstacles. The presented solution decomposes the task into subtasks that are solved by independent planners. This approach lead to an environment manipulation task planner, navigation motion planner and manipulation motion planner. The environment manipulation task planner decomposes the given task into navigation and manipulation tasks that are represented in a graph. Standard graph search techniques are then performed on the graph. The navigation motion planner first determines possible goal locations for manipulation configurations on the object chosen as the navigation goal before determining a collision free path. The manipulation

motion planner finds a goal location for a movable object. The goal location is chosen to be the closest to the original object location while not colliding with a navigation path from the global start location to the global goal location. However, this planner requires global world knowledge for planning and can not guarantee optimality. It is therefore not applicable to the domain discussed in this work.

In [23], Stilman presented a planner that solved a subclass of NAMO problems termed LP_1 where disconnected components of free-space could be connected independently by moving a single obstacle. This approach reduced the search space of NAMO by considering the difficulty of the navigation task rather than the dimensionality of the space. By formulating LP_1 problems as a graph of disjoint free space components, a resolution complete solution was found using a heuristic planner. The planner was able to solve the difficult problems presented in [2] and was successfully implemented on the humanoid robot HRP-2 [25]. Further work considered the correlated motions of multiple objects [24, 16] and presented a probabilistically complete algorithm for NAMO domains [28].

Li [13] constructed an autonomous system which combined moving objects and leaping over obstacles with other high-level behaviors using a unified planning strategy. However, these methods solved NAMO given complete knowledge about the environment. Furthermore, instead of aiming for global optimality, heuristics were used in order to find a feasible solution.

Wu [4] presented the first extension to the work of NAMO in known environments and introduced a planner that could solve NAMO in Unknown Environments. However the domain was restricted to push actions for obstacle manipulation, and obstacles were limited to rectangular shapes. While this planning algorithm was optimized for performance, it did not guarantee optimal decision making given available information.

Kakiuchi e.t all [7] has demonstrated a robot operating in an environment in

which the robot can manipulate obstacles to reach the goal while not having prior information about the obstacles. The robot only used onboard sensors to perceive it's environment. Movability of objects was detected based on the execution of a push action on the object and observing the result. However, this work was mainly focused on the perception aspects of the task. The actual planning within the domain was limited. The robot first tried to find a path to the goal while avoiding all the obstacles. If such as path is found then the plan is executed. If, however, no plan avoiding all the obstacles is found, the robot tries to push the object closest to it and re-checks if a path can be found. No actual planning incorporating the obstacles was performed. Consequently optimality can not be guaranteed for the robots actions.

2.2 Motion Planning in unknown environments

In more general approaches to motion planning, LaValle [12] presented a game-theoretic framework for robot motion in uncertain environments. Pirjanian [18] introduced many approaches to formulating the motion planning problem as action selection and also presented an implementation of Multiple Objective Action Selection for robot navigation [19]. By defining objective functions for different subgoals, Pareto optimal actions were calculated to determine "good enough" action for the current state. Although these methods are promising directions for decision making under uncertainty, it remains challenging to model problems within changeable environment configurations and maintain guarantees on the optimality of each action.

The D* algorithm [21] [22] incrementally searches paths in partially known environments by propagating the cost evaluated from the previous state to the new state. Thus, repeated replanning can be avoided without losing optimality. Koenig [9] introduced a rather less complicated algorithm, D* Lite, which only recomputes costs relevant to new information. The NAMO domain does not permit direct application of D* since the robot not only detects new obstacles, but also needs to manipulate

them. Appropriate extension of the algorithm would require a significant reformulation of the algorithm due to the drastically increased number of degrees of freedom. D* Lite can also not directly serve for the simple navigation planning portion in this domain since the robot may need to navigate to a grasping configuration, potentially at a significantly different configuration than the goal. This would change the root of the D* Lite search tree, making it inefficient [9]. However, D^* Lite can be made applicable to the NAMO domain as discussed below by keeping track of multiple search trees. This is implemented in the algorithms presented in this work.

Koenig [8] also established a series of techniques for goal-directed motion in the presence of incomplete information. This work suggested applying agent-centered search methods to minimize the cost of planning as well as plan execution. Koenig also used partially observable Markov decision process (POMDP) [1] to enhance the reliability of planning with incomplete information. POMDPs maintain and update a probabilistic model to minimize the cost of plan execution. Yet, such work was restricted to planning solutions that do not change the environment. It is expected that future development of the presented work will integrate these methods to handle environments with greater uncertainty.

One of the most successful algorithms in planning for unknown environments is the Bug family studied by Lumelski [14]. Bug algorithm provide complete solution to path planning towards a global goal based on local information. They also guarantee bounds on path length. Variants of the Bug algorithm [15] utilized different optimizations strategies such as reducing the length of the path or the information needed. However, the Bug family did not handle reconfiguration of environments and sacrificed optimality for completeness and planning efficiency. Inevitably, the resulting solutions had higher cost. In contrast to the limited memory used by bug algorithms, the method in this work incrementally constructs a complete model of the observed environment and makes the optimal decisions assuming that unknown

space is free.

CHAPTER III

PROBLEM FORMULATION

Navigation Among Movable Obstacles in Unknown Environments is defined as follows. A Robot R and a planar workspace containing a set $\mathcal{O}$ of static obstacles and a set $\mathcal{M}$ of movable obstacles are given. A configuration of the world, q_W, is defined by the position of the robot q_r and the position of each movable obstacle $q_i(M_i \in \mathcal{M})$. It is assumed that both the robot and the obstacles are rigid bodies that occupy a set of points in the workspace at any given time. Let G be the set of all points in the workspace. The following definitions define sets to indicate the occupied points:

$$R(t) = \{x \in G | x \text{ is occupied by } R \text{ at time } t\}$$

$$M_i(t) = \{x \in G | x \text{ is occupied by } M_i \in \mathcal{M} \text{ at time } t\}$$

$$O(t) = \{x \in G | x \text{ is occupied by any } O_i \in \mathcal{O} \text{ at time } t\}$$

The NAMO task can now be forumlated. The starting configurations for the robot and the movable obstacles are defined by $q_W^s = q_r^s, q_0^s \ldots q_n^s$. The goal is a target configuration for the robot q_r^g. While the algorithm is given q_r^s and q_r^g, it is not given any information about the dimensions, configurations or movability parameters for any of the obstacles. This information must be gathered online through limited sensing.

The NAMO task is to find a sequence of collision-free actions by the robot that results in the configuration q_r^g for the robot. The sequence of actions must ensure that at all times, t, the sets of points occupied by the robot, $R(t)$, each movable obstacle $M_i(t)$ and the static obstacles O do not intersect. Sensing information becomes available after each action execution.

3.1 Action Specification

The NAMO domain allows two types of robot action primitives: navigation, specified by the set $\mathcal{A}_N$, and manipulation, $\mathcal{A}_M$. Both action primitives are defined by paths τ_r of fixed length for the robot from a start configuration $\tau_r(t_s)$ to an end configuration $\tau_r(t_e)$. The time t at the beginning of the traversal of τ_r is noted as t_s and the end noted as t_e. Consequently Eq. 1 defines all the points in the workspace that the robot occupies during the execution of the action A.

$$\Gamma_r(A) = \bigcup_{t \in [t_s, t_e]} R(t) \tag{1}$$

Navigation Actions are simply path plans that displace the robot. Since the robot is allowed to change the environment, a sound plan must take into account the state of the world at a given time. Consequently, the free space of the robot at time t is therefore defined as follows:

$$F(t) = G \backslash (\bigcup_{M_i \in M} M_i(t) \cup O(t)) \tag{2}$$

This restricts the action $A_N \in \mathcal{A}_N$ to paths where $\Gamma_r(A_N) \cap F(t_s) \equiv \Gamma_r(A_N)$ holds. In other words the *swept volume*, $\Gamma_r(A_N)$, of action A_N does not intersect the obstacles.

In the case of manipulation actions there is an additional constraint on the starting point for the action defined by the relative configuration of the robot and one movable obstacle. Furthermore, a function $\tau_{Mi} \leftarrow \mathfrak{M}(\tau_r)$ is defined that maps the motion of the robot to a displacement of the obstacle M_i. In the implementation presented in this work, $\mathfrak{M}$ is simply a transformation of the robot path to the coordinate system of the object. This is consistent with grasp and displacement. The precise specification of $\mathfrak{M}$ is not critical to the algorithm or analysis. Alternative definitions of F can restrict A_M to pushing or other manipulation primitives.

Manipulation Actions are path plans that displace both the robot and obstacle M_i. In accordance with Γ_r, which specifies the workspace points occupied by the

robot during the execution of A_N or A_M, Γ_{M_i} denotes the points occupied by the obstacle M_i during the execution of A_M.

$$\Gamma_{M_i}(A_M) = \bigcup_{t \in [t_s, t_e]} M_i(t) \tag{3}$$

In contrast to navigation actions, $F(t)$ changes during the execution of manipulation actions. $A_M \in \mathcal{A}_M$ is therefore restricted to paths where equations (4)-(6) hold:

$$\forall t \in [t_s, t_e] : R(t) \cap F(t) \equiv R(t) \tag{4}$$

– The robot is intersection free.

$$\forall t \in [t_s, t_e] : (\forall M_j \neq M_i \in \mathcal{M} : M_i(t) \cap M_j(t) \equiv \emptyset) \tag{5}$$

– The manipulated obstacle M_i is intersection free with all other movable obstacles.

$$\forall t \in [t_s, t_e] : M_i(t) \cap O(t) \equiv \emptyset \tag{6}$$

– M_i is intersection free with all static obstacles.

3.2 Unknown Environments

Since the robot is only given partial environment information, it must be defined how the robot interprets its understanding of the world. Incomplete information comes from two possible sources at any time t:

- *$\mathcal{M}$ and $\mathcal{O}$ are incomplete or erroneous:*

  There may not be sufficient information to determine $\mathcal{M}$ and $\mathcal{O}$, leading to missing objects or assumptions that non-existent obstacles exist. *Incomplete* information comes from missing sensor data. *Erroneous* conclusions come from assumptions on movability before interaction.

- *Incomplete $M_i(t)$ and $O(t)$:*

  The set $M_i(t)$ for an $M_i \in \mathcal{M}$ may not be complete as the entire range of the object may not have been detected at time t. The same applies for obstacles $O_i \in \mathcal{O}$ and as such for the set $O(t)$.

In this study, the following minimal set of assumptions to handle the challenges of unknown environments are made:

1. *Obstacles are assumed movable:* An obstacle is movable unless a manipulation action failed on the obstacle.

2. *Free space:* Unknown space is assumed to be free of obstacles.

The first assumption enables an insert of newly detected obstacles into the set $\mathcal{M}$, and defines an explicit rule of when obstacles have to be removed from $\mathcal{M}$ and inserted into $\mathcal{O}$. The second assumption compensates for incompleteness of the sets $\mathcal{M}, \mathcal{O}, M_i(t)$ and $O(t)$. Following these assumptions the sets can be treated as correct at any given time t.

3.3 Action Sequences

A consecutive sequence of actions is denoted as a plan $\mathcal{P}$. The actions A in a plan $\mathcal{P}$ will be indexed from A^1 to A^n. The *swept volume* of $\mathcal{P}$ is defined by Eq. 7. It represents all the points occupied by the robot and manipulated obstacles.

$$S(\mathcal{P}) = \bigcup_{A \in p} \Gamma_r(A) \bigcup_{A_M \in p} \Gamma_{M_i}(A_M) \qquad (A_M \in \mathcal{A}_M) \tag{7}$$

In the following the robots configuration at time t is denoted as q_r^t. Further, the set $P(t)$ is defined to be the set of all sound plans $\mathcal{P}$ at time t that satisfy the following constraints:

1. transfer R from the robots current configuration q_r^t to its goal configuration q_r^g

2. satisfy the restrictions described in section 3.4

Let $P_{M_i}(t) \subseteq P(t)$ be the set consisting of sound plans having a non-zero sequence of manipulation actions on the movable obstacle M_i. Similarly, the set $P_{\text{-}}(t) \subseteq P(t)$ denotes the set of sound plans only consisting of navigation actions. A plan $\mathcal{P} \in P_{\text{-}}(t)$ will be denoted as pure navigation plan. This yields:

$$P(t) = \bigcup_{M_i \in M} P_{M_i}(t) \cup P_{\text{-}}(t) \tag{8}$$

3.4 Domain Restrictions

The domain is restricted as follows in this work:

3.4.1 Manipulation

The possible interactions with obstacles are limited to axis-aligned manipulations. Obstacles have to be grasped at the center of one of the axis-parallel sides.

3.4.2 Single obstacle

A plan $\mathcal{P}$ involves at most a single consecutive sequence of manipulation actions:

$$\begin{aligned} \forall x \in [1 \cdots n] : A^x \equiv A_N \vee \exists i, j \in [1, \cdots, n] : \\ i \leq x \leq j \Leftrightarrow A^x \equiv A_M \end{aligned} \tag{9}$$

This restriction is only in reference to a plan $\mathcal{P}$ given the current environment knowledge. Since the environment knowledge can change, the robot may manipulate multiple obstacles before reaching the goal configuration.

3.4.3 No obstacle interactions

Obstacles are solid and can only be moved by the robot.

3.4.4 Goal is always free

The points in the workspace occupied by a configuration q_r^g are always free of obstacles.

3.5 Cost Definition

A constant cost, $\mathcal{C}_M$, is assigned to manipulation action primitives, A_M, as well as a constant cost, $\mathcal{C}_N$, to navigation action primitives, A_N. The following inequality holds:

$$\mathcal{C}_M > \mathcal{C}_N \geq 0 \tag{10}$$

Following equation (9), let i be the index of the first manipulation action, j the index of the last manipulation action. If $\mathcal{P}$ does not include any manipulation actions $i, j = 0$. The cost of any plan $\mathcal{P} \in P(t)$ can be defined to be the number of navigation actions times the cost for navigation and the number of manipulation actions times the cost for a manipulation action.

$$\mathcal{C}(\mathcal{P}) = (i + n - j)\mathcal{C}_N + (j - i)\mathcal{C}_M \tag{11}$$

3.6 Optimality Definition

Let $\mathcal{P}^*$ be the plan with the least cost.

$$\mathcal{P}^* = argmin_{\mathcal{P} \in P(t)}(\mathcal{C}(\mathcal{P})) \tag{12}$$

Due to assumption 3.4.2 and a simple rewrite of equation (11) any $\mathcal{P} \in P(t)$ can be split into three parts, navigating to the obstacle, manipulating the obstacle and navigating to the goal. Equation (11) can therefore be rewritten as

$$\mathcal{C}(\mathcal{P}) = c_1 + c_2 + c_3 \tag{13}$$

with

$$c_1 = iC_N \tag{14}$$

$$c_2 = (j - i)C_M \tag{15}$$

$$c_3 = (n - j)C_N \tag{16}$$

and substituting (13) into (12) yields

$$\mathcal{P}^* = argmin_{\mathcal{P} \in P}(c_1 + c_2 + c_3) \tag{17}$$

If $\mathcal{P}$ does not include a manipulation action c_1 and c_2 evaluate to zero. A visualization of the costs associated with a plan can be seen in Fig. 2.

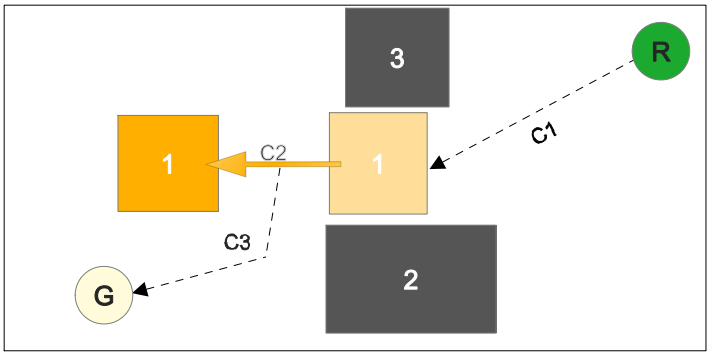


Figure 2: The three phases of a plan

Further, let $\mathcal{P}^*_{M_i} \in P_{M_i}(t)$ be the optimal plan with at least one manipulation action on the obstacle M_i.

$$\mathcal{P}^*{}_{M_i} = argmin_{\mathcal{P} \in P_{M_i}(t)}(c_1 + c_2 + c_3) \tag{18}$$

3.7 Openings

In the following chapters the term *new opening* will be used. While a detailed discussion about this is proposed until chapter 6, for clarity, the general definition will

be given here. The definition of a *new opening* is based on the notion of a *homotopic* plans, which will be defined first.

Definition 1. *Two pure navigation plans $\mathcal{P}_1, \mathcal{P}_2 \in P_{\_}(t)$ are homotopic if and only if there exists no known obstacle in the area enclosed by the paths that the robot traverses if executing the plans. Otherwise they are ahomotopic.*

This definition is adopted from [5]. Based on this definition the definition of a new opening can be given.

Definition 2. *A manipulation action created a new opening if the set $P_{\_}(t')$ at time t' after the execution of the manipulation action has at least one plan $\mathcal{P}$ that is ahomotopic to all the plans in the set $P_{\_}(t)$ at time t prior to the execution of the manipulation action.*

CHAPTER IV

ALGORITHMS

In this algorithms, the space is represented by a grid. Each of the sets of occupied points is defined by the set of grid cells that contain at least one point.

The optimality definition provided in (12) can be expanded by using the set definition (8). This yields that $\mathcal{P}^*$ is the plan $\mathcal{P} \in \{\mathcal{P}^*_{M_1}, \cdots, \mathcal{P}^*_{M_n}, argmin_{\mathcal{P} \in P\_}(C(\mathcal{P}))\}$ with minimal cost. $\mathcal{P}^*$ can therefore easily be determined if all those plans are known. The algorithms explained in this work follow these intuition. In the following the calculation of $\mathcal{P}^*_{M_i}$ will be referred to as performing an obstacle evaluation on M_i.

Further the terminology *constructing a plan* is used to indicate that a plan is actually constructed, e.g. actual navigation actions are determined to exactly calculate c_1 and c_3. This is in contrast to the *estimation* of those costs, where no actual plan is constructed.

This chapter first describes the simplifications that are going to be used for the algorithm explanations. Next the *Baseline* algorithm is presented, a naïve algorithm that optimally solves NAMO in Unknown Environments. Finally this chapter is providing a detailed description of the *Optimized* algorithm that proves to drastically reduce the runtime while still guaranteeing optimality. The optimality proofs for both algorithms will be given in Chapter 5.

4.1 Algorithm Simplifications

For clarity, some details of the algorithms are omitted that do not represent the main part of the algorithms. This allows for the pseudocodes and diagrams to be more detailed on the core parts of this work. In the following those simplifications will be stated.

4.1.1 Static Objects

Static objects will not be explicitly be mentioned. All cells that correspond to a detected static object are marked as blocking robot and object motion in future plans.

4.1.2 Multiple Grasping points

In Chapter 3.4.1 it was defined that obstacles have to be grasped on one of the axis parallel sides of the object. This can yield between zero and four possible grasping points per obstacle depending on its shape. It can not generally be assumed that the grasping point currently closet to the robot has to be chosen in order to find $\mathcal{P}^*_{M_i}$. It is therefore necessary to evaluate all valid grasping points.

The actual implementations of the algorithms described below are therefore evaluating plans for all possible grasping points in parallel in different threads. If one thread returns, inter thread communication is used to kill threads that can, based on its calculations, not yield a plan with lower cost anymore (for details on this estimation see section 4.3.2). This process will not be reflected in the pseudocode or algorithm descriptions.

4.1.3 Navigation Planner

For simplicity, A^* is assumed as a navigation planner for the algorithm as well for the proofs. The actual implementation of the algorithm however is based on a variation of D^* Lite. Since the use of D^* Lite is not essential to the concepts presented in this work and does not affect the optimality aspects, is its discussion postponed to Chapter 7.

4.2 Baseline

The naive solution to optimal NAMO in Unknown Environments is to calculate plans for all possible actions on all known objects once any change in the environment is

Algorithm 1 BASELINE(q_r^s, q_r^g)

```
1: R ⇐ q_r^s;
2: M ⇐ ∅;
3: P* ⇐ A*(q_r^s, q_r^g);
4: while R ≠ q_r^g do
5:    O_new ⇐GET-NEW-INFORMATION();
6:    if O_new ≠ ∅ then
7:       M = M ∪ O_new;
8:       for each M_i ∈ M do
9:          P ⇐ EVALUATE-ACTIONS(M_i);
10:         if C(P) < C(P*) then
11:            P* = P;
12:         end if
13:      end for
14:      P_avoid = A*(q_r^t, q_r^g);
15:      if C(P_avoid) < C(P_opt) then
16:         P* = P_avoid;
17:      end if
18:   end if
19:   R ⇐ Next step in P*;
20: end while
```

detected. $\mathcal{P}^*$ can than simply be chosen to be the plan with minimum cost. The Baseline algorithm basically implements this intuition, with just a small modification on the navigation planning. Not all plans with equivalent manipulation actions are constructed. Rather just one plan is constructed having these manipulation actions whereby c_1 and c_3 are calculated based on the navigation actions returned by the navigation planner using on A^*.

The algorithm is initialized with a direct path to the goal and re-planning is performed once environment changes are detected.

This approach is visualized in fig. 3 and outlined in Algorithms 1 and 2.

4.3 Optimized algorithm

In order to gain scalability for big maps with a high number of objects, the optimized algorithm alters the Baseline algorithm in three main aspects.

The first optimization step drops the necessity to re-evaluate the set $P(t)$ of all

Algorithm 2 EVALUATE-ACTIONS(M_i)

1: $\mathcal{P}^*_{M_i}, closedList, Q \Leftarrow \emptyset$
2: $\mathcal{C}(\mathcal{P}^*_{M_i}) = \infty$;
3: $\mathcal{P}_{to\_obstacle}$ =A$^*(q^t_r, q^t_{M_i})$;
4: INSERT $(Q, \mathcal{P}^*_{M_i})$;
5: INSERT $(closedList, q^s_{M_i})$;
6: **while** $Q \neq \emptyset$ **do**
7: $\quad \mathcal{P}_x \Leftarrow$POP$(Q)$;
8: $\quad$ **for** d in $\{left, right, up, down\}$ **do**
9: $\quad\quad q^{tmp}_{M_i} \Leftarrow$GET_POSITION$(M_i, P_x) + d$;
10: $\quad\quad$ **if** !HAS_ELEMENT$(closedList, q^{tmp}_{M_i})$ **then**
11: $\quad\quad\quad$ INSERT$(closedList, q^{tmp}_{M_i})$;
12: $\quad\quad\quad$ APPEND$(\mathcal{P}_x, d)$;
13: $\quad\quad\quad \mathcal{P}_{to\_goal}$ =A$^*(q^t_{M_i}, q^g_r)$;
14: $\quad\quad\quad \mathcal{P} = \mathcal{P}_{to\_obstacle} + \mathcal{P}_x + \mathcal{P}_{to\_goal}$;
15: $\quad\quad\quad$ **if** $\mathcal{C}(\mathcal{P}) < \mathcal{C}(\mathcal{P}^*_{M_i})$ **then**
16: $\quad\quad\quad\quad \mathcal{P}^*_{M_i} = \mathcal{P}$
17: $\quad\quad\quad$ **end if**
18: $\quad\quad\quad$ INSERT$(Q, \mathcal{P}_x)$;
19: $\quad\quad$ **end if**
20: $\quad$ **end for**
21: **end while**
22: return $\mathcal{P}^*_{M_i}$;

sound plans (see section 3.3) on every environment update. The second optimization steps reduces the calculations performed by an obstacle evaluation. Finally, the third optimization step potentially eliminates entire sets $P_{M_i}(t)$ of plans with a non-zero manipulation sequence on M_i (see section 5) from equation (8).

Similar to the *baseline* algorithm, plans with equal manipulation actions on an object M_i are not evaluated for all possible c_1 and c_3. These costs are determined once through a navigation plan computed with A^* used as the navigation planner.

4.3.1 Recalculation triggering

In contrast to the *Baseline* algorithm re-planning is not automatically performed upon the detection of new objects or updated object information in the *optimized* algorithm.

Algorithm 3 OPTIMIZED(q_r^s, q_r^g)

$R \Leftarrow q_r^s$;
$minCost, euclidianCost \Leftarrow \emptyset$;
$mC\_pt, eC\_pt = 0$;
$\mathcal{P}^* \Leftarrow A^*(q_r^s, q_r^g)$;
while $R \neq q_r^g$ **do**
  $\mathcal{O}_{new} \Leftarrow \mathcal{O}_{new} \cup$ GET-NEW-INFORMATION();
  if $S(\mathcal{P}^*) \cap \mathcal{O}_{new} \neq \emptyset$ **then**
    $\mathcal{P}^* \Leftarrow A^*(q_r^t, q_r^g)$;
    for each $M_i \in \mathcal{O}_{new}$ **do**
      UPDATE($euclidianCost, M_i$);
    end for
    $\mathcal{P}_{next}$=GET-NEXT(mC_pt, eC_pt);
    while $\mathcal{C}(\mathcal{P}^*) \geq \mathcal{C}(\mathcal{P}_{next})$ **do**
      if mC =true or $\mathcal{P}_{next} \notin minCost$ **then**
        $\mathcal{P}$=OPT-EVALUATE-ACTION($\mathcal{P}_{next}$,$\mathcal{P}^*$);
        UPDATE($minCost, [M_i, Cost_l(M_i)]$);
        if $\mathcal{C}(\mathcal{P}) < \mathcal{C}(\mathcal{P}^*)$ **then**
          $\mathcal{P}^* = \mathcal{P}$;
        end if
        $\mathcal{P}_{next}$ = GET-NEXT(mC_pt, eC_pt);
      end if
    end while
    $\mathcal{O}_{new} \Leftarrow \emptyset$;
  end if
  $mC\_pt, eC\_pt = 0$;
  if Next step in $\mathcal{P}^* \equiv A_M$ **then**
    $minCost \Leftarrow \emptyset$
  end if
  $R \Leftarrow$ Next step in $\mathcal{P}^*$;
end while

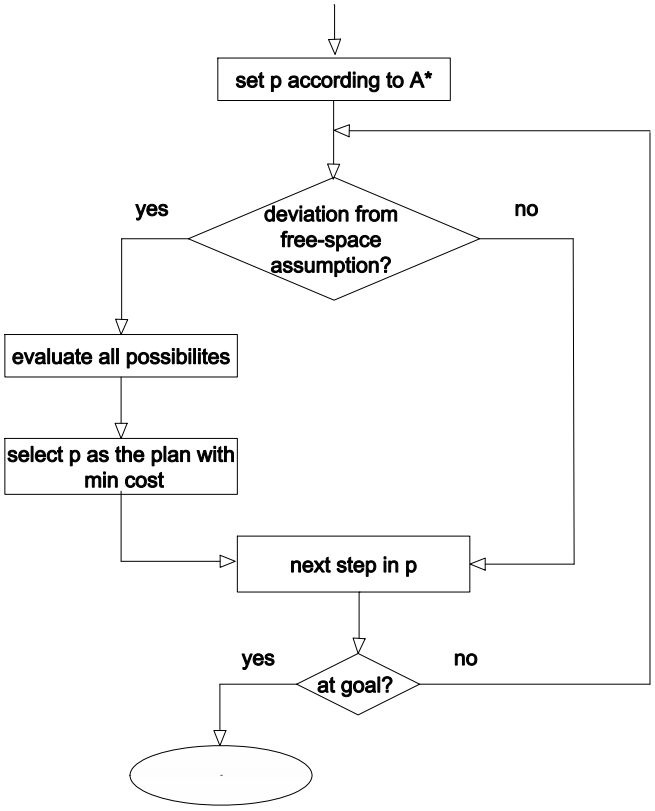


Figure 3: Flow chart of baseline algorithm

In the *optmized* algorithm re-planning is only triggered if the currently executed plan $\mathcal{P}$ is no longer intersection free with the newly detected environment information. If $\mathcal{P}$ is intersected it is invalid. Let O_{new} be the set of all points which are newly detected as occupied:

$$O_{new} = \{x \in G | x \text{ is occupied but was assumed free at the calculation time of } \mathcal{P}^* \}$$

Re-planning is then performed according the following constraint:

$$S(\mathcal{P}^*) \cap O_{new} \neq \emptyset \Rightarrow \mathcal{P}^* \notin P \Rightarrow \textit{re-plan} \tag{19}$$

This optimization step can be seen in line 6, 7 and 23 in algorithm 3.

Algorithm 4 GET-NEXT(mC_pt, eC_pt)

1: mC = false;
2: $\mathcal{P}_{mC} = minCost(mC\_p)$;
3: $\mathcal{P}_{eC} = euclidianCost(eC\_p)$;
4: **if** $\mathcal{C}(\mathcal{P}_{mC}) \leq \mathcal{C}(\mathcal{P}_{eC})$ **then**
5: $\quad \mathcal{P}_{next} = \mathcal{P}_{mC}$;
6: $\quad mC\_pt$++;
7: $\quad mC$=true;
8: **else**
9: $\quad \mathcal{P}_{next} = \mathcal{P}_{eC}$;
10: $\quad eC\_pt$++;
11: $\quad mC$ =false;
12: **end if**
13: return $(mC, \mathcal{P}_{next})$;

4.3.2 Limit Navigation Planner calls

In the following discussion $\mathcal{P}^*_{tmp}$ refers to the minimum cost plan determined through calculations in the current map configuration so far. In order to reduce calculations necessary in determining $\mathcal{P}^*_{M_i}$, are plans for a valid sequence of manipulation actions seq on M_i omitted, if seq

(a) is not creating a new opening

(b) the estimated cost $\mathcal{C}_{est}(\mathcal{P})$ is exceeding $\mathcal{C}(\mathcal{P}^*_{tmp})$

where

$$\mathcal{C}_{est}(\mathcal{P}) = c_1 + |seq|\mathcal{C}_N + euclidianCost \tag{20}$$

with

$$euclidianCost = \lceil \frac{|q^t_r - q^g_r|}{dis} \rceil \mathcal{C}_N \tag{21}$$

whereby dis represents the distance a single move action primitives translates the robot.

The plan $\mathcal{P}^*_{tmp}$ therefore functions as an upper bound and is initially set to be a plan avoiding all obstacles. It is updated constantly if a new plan with lower cost is found. This can be seen in line 8 and 17-19 of algorithm 3.

If condition (b) is reached, no more plans with *seq* as a beginning manipulation sequence are considered.

Part (a) can be seen in line 25 and part (b) in lines 22-24 of algorithm 5.

New openings can only occur in space directly adjacent to the currently manipulated obstacle, which is shown in section 5.2.2. The definition of a new opening is given above in definition 2. How opening detection is implemented in the code will be discussed in Chapter 6.

4.3.2.1 Special case: no initial $\mathcal{P}^*_{tmp}$

A special case can occur if no collision free path avoiding all the object can be found, yielding $\mathcal{C}(\mathcal{P}^*_{tmp}) = \mathcal{C}(\mathcal{P}_{M_i}) \equiv \infty$ initially, and as such not providing an upper bound for evaluating sequences. However, if no bound can be provided all possible manipulations for an obstacle have to be evaluated until

- a valid plan was found (and as such providing an upper bound) or
- the entire search-space for manipulations on that obstacle was searched.

As can be easily imagined can this result in an intractable amount of calculations. For example, if an obstacle is evaluated which, however manipulated, can not make the goal accessible is chosen to be evaluated first. This object will still have to be evaluated for all possible manipulations. For a large map this search-space could not be searched in reasonable time and as such practically prevent the robot from reaching the goal. This can even occur when potentially first evaluating a different obstacle could have provided a path to the goal with a limited amount of computations.

A basic solution to this problem is to perform all obstacle evaluations on the known movable obstacles in parallel. The proposed algorithm implements a variation of this solution.

First, the observation can be made that it is possible to determine which objects are not effecting the robots ability to reach the goal. This, for example, can be done by setting the space occupied by an obstacle temporary to be free space. It can now be verified if a path to the goal can be found. If no path can be found then this object is considered *non-blocking*, however if a path can be found then it is a *blocking* object. This process is repeated for each object.

The algorithm now iterates over the list of *blocking* objects and in each iteration, increasing the maximum number of allowed manipulation actions on each object. This is similar to a breath-first search. Upon the detection of a new opening, it is attempted to construct a plan to the goal. If such a plan can successfully be constructed, $\mathcal{P}^*_{tmp}$ is set to be equal to this plan. Otherwise the procedure is continues. Once a path to the goal has been found $\mathcal{P}^*_{tmp}$ can be set to a value smaller than infinity and the *optimized* algorithm can continue its execution as described above. This special case is not outlined in the pseudo-code or diagram.

4.3.3 Reduce candidate objects

Not every $\mathcal{P}^*_{M_i}$ is determined but rather only those where a computation seems promising. To achieve this two sorted lists *minCost* and *euclidianCost* are maintained. Both lists contain underestimates of the true costs to reach the goal if a plan with a non-zero sequence of manipulation actions on M_i is constructed.

Every element in the list *minCost* is a tuple associating an obstacle $M_i \in M$ with the smallest $\mathcal{C}_p(\mathcal{P})$ determined by the latest evaluation of M_i. The *minCost* list therefore represents underestimates for any $\mathcal{C}(\mathcal{P})$ having a non zero sequence of manipulation actions on M_i for most times. However, elements in this list are not

guaranteed to represent a lower bound if free-space has been created, eg. by moving an object, that was assumed blocked at the time an entry was inserted into the list.

Updating the *minCost* list in case free-space creation is tedious, since it is not trivial to determine which entries in the list are actually affected by this change. The algorithm therefore entirely invalidates the *minCost* list once free-space has been created and lazily recover elements of the list if necessary. For recovering purposes are a second list, *euclidianCost*, is maintained. This list again consists of tuples but associates all movable objects encountered so far with an underestimate of just c_3. The estimate can be calculated using equation (25). Other space-representation specific quantities are also possible. The only requirement on this estimate is that it is an under estimate and is independent of observed obstacles (e.g. assumes free-space even in known parts of the world). The *euclidanCost* list is never invalidated, however has update operations. Update operations are *insert*, for newly detected obstacles, and *obstacle update* for the cases when more information about an already known obstacle becomes available (it increases in size) or an object is moved. This is necessary because those cases could affect the grasping point and as such the estimate for c_3.

Both lists are sorted in ascending order.

While both lists contain underestimated, the *minCost* list usually provides tighter bounds.

Upon recalculation triggering both lists are traversed in parallel until the next smallest cost associated with the next entry exceeds the cost of $\mathcal{P}^*_{tmp}$ which is set and updated as described in chapter 4.3.2.

Traversal is done in such a way that the next element is chosen to be the smallest of the minimum of the two lists. However if this tuple belongs to *euclidianCost* and the associated obstacle is also referenced in *minCost* the entry is ignored and traversal continued (since a tighter bound exists). If the next smallest element either

- belongs to *minCost* or
- belongs to *euclidianCost* but is not referenced in *minCost*

the associated obstacle is evaluated and *minCost* and $\mathcal{P}^*_{tmp}$ are updated if applicable.

Figure 4 summarizes this optimization step in an UML diagram and fig 17 demonstrates an easy example.

The update operation on the *euclidianCost* list can be seen in lines 9-11 of algorithm 3. The method of choosing the next element can be found in line 12 of algorithm 3 and in detail in algorithm 4. In algorithm 5, the entries for the *minCost* list is calculated through the lines 29-31. This value is then used in line 16 of algorithm 3 to update *minCost*.

Algorithm 5 OPT-EVALUATE-ACTION(M_i, $\mathcal{P}^*$)

$\mathcal{P}^*_{M_i}, Q, closedList \Leftarrow \emptyset$;
$\mathcal{C}_l(M_i) \leftarrow \infty$;
$\mathcal{P}_{to\_obstacle} = A^*(q^t_r, q^t_{M_i})$;
if $\mathcal{P}_{to\_obstacle}$ NOT valid **then**
  return $failure$
end if
$\mathcal{C}(\mathcal{P}^*_{M_i}) = \infty$;
INSERT$(Q, \mathcal{P}^*_{M_i})$;
INSERT $(closedList, q^s_{M_i})$;
while $Q \neq \emptyset$ **do**
  $\mathcal{P}_x \Leftarrow$POP(Q);
  for d in $\{left, right, up, down\}$ **do**
    $q^{tmp}_{M_i} \Leftarrow$GET_POSITION$(M_i, \mathcal{P}_x) + d$;
    if !HAS_ELEMENT$(closedList, q^{tmp}_{M_i})$ **then**
      skip;
    end if
    INSERT$(closedList, q^{tmp}_{M_i})$;
    if $q^{tmp}_{M_i}$ causing robot or obstacle collision **then**
      skip;
    end if
    APPEND$(\mathcal{P}_x, d)$;
    if $\mathcal{C}_{est}(\mathcal{P}_x) \geq \mathcal{C}(\mathcal{P}^*_{M_i})$ **or** $\mathcal{C}_{est}(\mathcal{P}_x) \geq \mathcal{C}(\mathcal{P}^*)$ **then**
      skip;
    end if
    if new opening was created **then**
      $\mathcal{P}_{to\_goal} = A^*(q^{tmp}_{M_i}, q^g_r)$;
      if $\mathcal{P}_{to\_goal}$ is valid **then**
        $\mathcal{P} = \mathcal{P}_{to\_obstacle} + \mathcal{P}_x + \mathcal{P}_{to\_goal}$;
        if $\mathcal{C}(\mathcal{P}_x) + \mathcal{C}(\mathcal{P}_{to\_goal}) < \mathcal{C}_l(M_i)$ **then**
          $\mathcal{C}_l(M_i) \leftarrow \mathcal{C}(\mathcal{P}_x) + \mathcal{C}(\mathcal{P}_{to\_goal})$;
        end if
        if $\mathcal{C}(\mathcal{P}) < \mathcal{C}(\mathcal{P}^*_{M_i})$ **then**
          $\mathcal{P}^*_{M_i} = \mathcal{P}$;
        end if
      end if
    end if
    INSERT$(Q, \mathcal{P}_x)$;
  end for
end while
if $\mathcal{P}^*_{M_i}$ was set **then**
  return $(\mathcal{P}^*_{M_i}, \mathcal{C}_l(M_i))$;
else
  return $\emptyset$;
end if

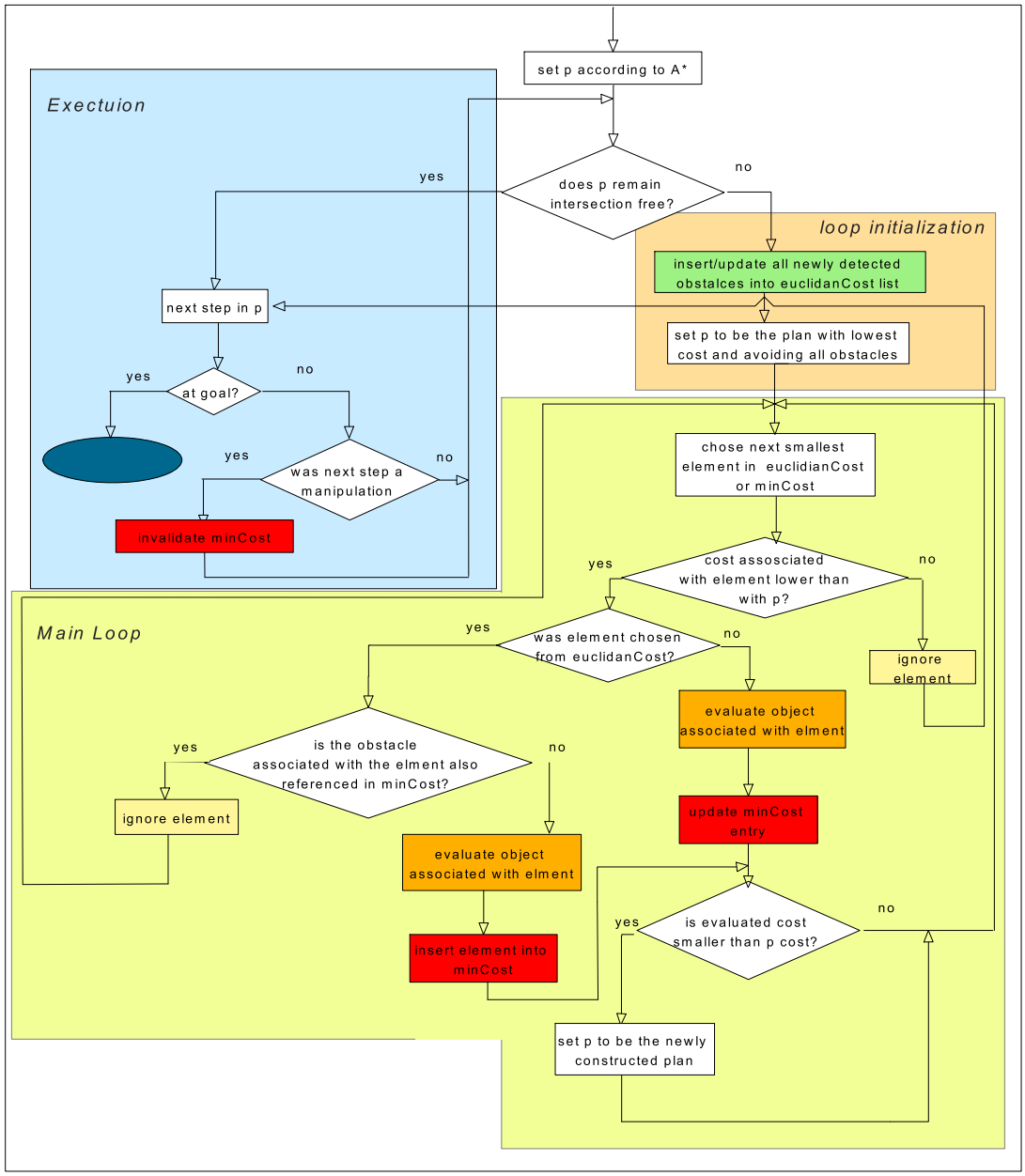
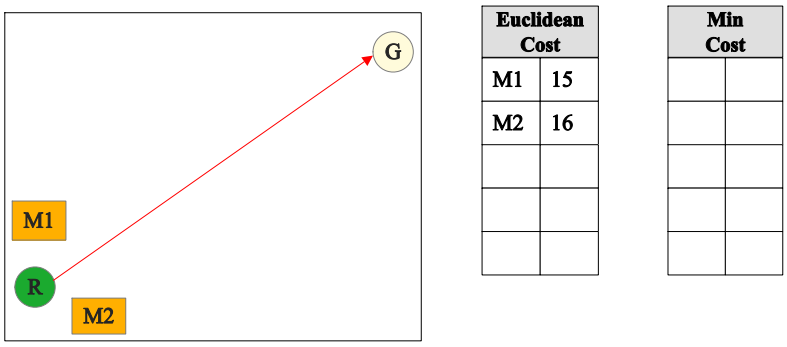
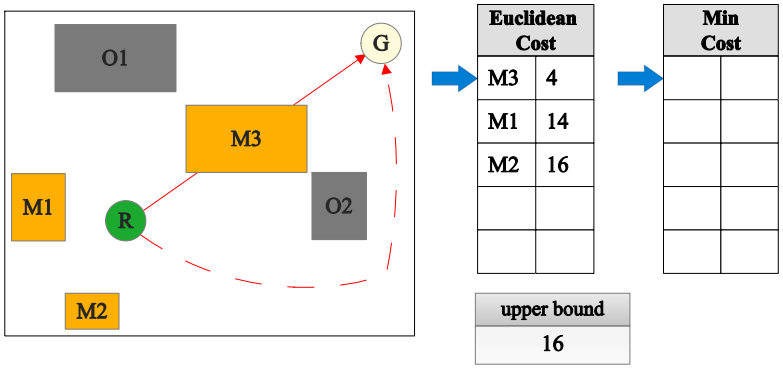


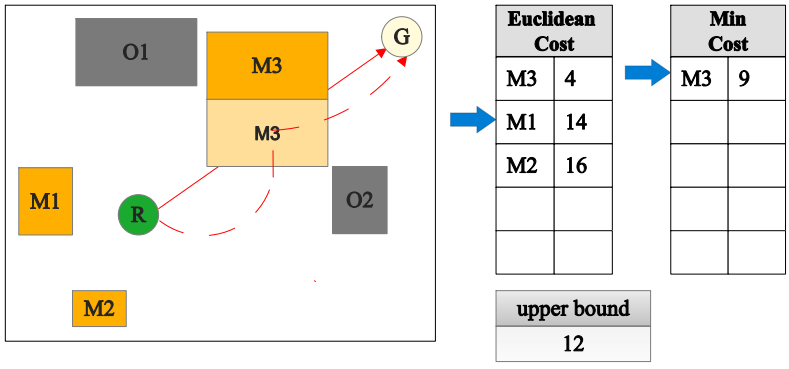
Figure 4: Flow chart for optimization step 4.3.3. Green: *euclidianCost* operations, red: *minCost* operations, orange: actual evaluation points, yellow: skipping points.



(a) No calculation necessary, obstacles only saved in EuclidianCost list



(b) Plan intersected, a path avoiding all obstacles could be found but entry with lower cost estimate in one of the lists (EuclidanCost), traversal started. The *euclideanCost* for the obstacle $M1$ was updated as new information of $M1$ became available.



(c) Evaluation of $M3$ since it is the next smallest entry of both lists and in *EuclideanCost* but not referenced in *MinCost*. The upper bound is updated and traversal stops. $M2$ and $M1$ will not be evaluated.

Figure 5: Example of optimization step 4.3.3

CHAPTER V

OPTIMALITY

As required in Chapter 3 has $\mathcal{P} = \mathcal{P}^*$ to hold at each time t for the currently executed plan $\mathcal{P}$. Consequently, following equation (17) the sum of c_1, c_2 and c_3 has to be minimal for $\mathcal{P}$. This includes the cases of having c_1 and c_2 set to zero, yielding a plan without obstacle interaction as defined in chapter 3.4.2.

This chapter will provide optimality proofs for the *baseline* as well as the *optimized* algorithm.

5.1 Baseline

In the following optimality for the *Baseline* algorithm will be shown.

Lemma 1. *The Baseline algorithm described in 4.2 is optimal.*

Proof. The *Baseline* algorithm is initialized with an A^* search from q_r^s to q_r^g using the *euclideanCost* (equation (25)) as the heuristic. This heuristic is an admissible heuristic. Following assumption 2 (assumed free-space in unknown environment) is the algorithm therefore initialized with $\mathcal{P}^*$.

For all plans having equal manipulation actions on one object, only the plans with c_1 and c_3 determined through a navigation plan returned by A^* are considered. This is valid since following equation (13) the cost is the summation of the positives value c_1, c_2 and c_3. For plans with equal manipulation actions c_2 is equal. The costs c_1 and c_3 are minimized if determined through A^* using and admissible heuristic as is done here.

Upon the detection of any new environment information interfering with the assumption stated in chapter 2 (free-space assumption) the algorithm evaluates all

sequences of all possible manipulations for each known object. c_1 and c_3 are again based on a navigation plan returned by an A^* search. Further is the case of c_1 and c_2 being equal to zero - yielding a direct path to the goal without obstacle interaction- based on the A^* search evaluated.

Because of the assumptions described in chapter 3.4.2 (a plan involves at most a single consecutive sequence of manipulation actions) and 3.4.3 (obstacles are only moved by the robot) does this cover all valid $\mathcal{P} \in P$ with the exception of plans with equivalent obstacle interaction but differing sequences of moving to the obstacle and or to the goal. However as described above these sequences can not have lower cost than the ones found through the A^* search.

The costs for each plan are than calculated according to equation (13). Finally, $\mathcal{P}$ is determined according to equation (17) and as such is equal to $\mathcal{P}^*$. This is valid for each time t since reevaluation is performed upon any change of the environment and consequently any possible change on the set P.

□

5.2 Optimized algorithm

In the following discussion it will be shown that optimality can still be guaranteed for the *optimized algorithm* described in chapter 4.3. This will be done by showing that none of the optimization steps are affecting the property of $\mathcal{P} = \mathcal{P}^*$ at any time t.

The time of the calculation of the currently executed plan $\mathcal{P}$ will be denoted as t_{calc}. The current time as t_{cur}. Similarly, if necessary, will P_{calc} be used to denote the set P at time t_{calc}. P_{cur} and $\mathcal{P}^*_{cur}$ will denote P and $\mathcal{P}^*$ at the current time. The term *occupied* a grid field will be used to indicate that a certain plan $\mathcal{P}_x \in P$ includes either a manipulation action or navigation action that causes the robot or an obstacle to enter a grid field.

5.2.1 Recalculation triggering

It is now shown that the optimization step described in chapter 4.3.1 of delayed re-planning is valid. This will be done by demonstrating that if $\mathcal{P} \equiv \mathcal{P}^*$ at t_{calc} and newly detected obstacles information do not intersect with $\mathcal{P}$ than $\mathcal{P} \equiv \mathcal{P}^*_{cur}$. Lemma 2 will first state that this holds for the detection of entirely new obstacles. Lemma 3 will than show that it also holds for updated size information of previously known obstacles.

Lemma 2. *A newly detected object M_i which does not intersect with any part of the current optimal plan $\mathcal{P}$ can not affect the optimality of $\mathcal{P}$. This is $M_i(t) \cap S(\mathcal{P}) \equiv \emptyset \Rightarrow \mathcal{P} \equiv \mathcal{P}^*_{calc}$.*

Proof. Due to assumption 2 (free-space assumption) every $\mathcal{P}_x \in P_{calc}$ is calculated assuming free-space at the grid fields which are now occupied by obstacles. Consequently every $\mathcal{P}_x \in P_{calc}$ *occuping* the newly as occupied detected grid cells becomes invalid and is not an element of P_{cur}. Let $P_{inv} = \{\mathcal{P} | \mathcal{P} \in P_{calc} \wedge S(\mathcal{P}) \cap M_i(t) \neq \emptyset\}$ be the set of all such plans.

Let $P_{new} = \{\mathcal{P} | \mathcal{P} \in P(t) \wedge \mathcal{P} \notin P_{calc}\}$ define the set of all new valid plans that can be constructed, yielding $P_{cur} = (P_{calc} \setminus P_{inv}) \cup P_{new}$. $\mathcal{P} \notin P_{inv}$ following the assumption given in the lemma. In the following it is shown that every $\mathcal{P}_x \in P_{new}$ has to have higher cost than $\mathcal{P}$, yielding that $\mathcal{P} \equiv \mathcal{P}^*_{cur}$.

Prove by contradiction. First, any $\mathcal{P}_x \in P_{new}$ has to have a non-zero sequence of manipulation actions on the newly detected obstacle, otherwise it would have already been an element of P_{calc}.

Assume $\exists \mathcal{P}_{new} \in P_{new} : \mathcal{C}(p_{new}) \leq \mathcal{C}(p)$. There has to be a $\mathcal{P}_x \in P_{calc}$ similar to $\mathcal{P}_{new} \in P_{new}$ just having navigation actions instead of manipulation actions. This $\mathcal{P}_x$ has to exists because of the free-space assumption in chapter 2. Also due to inequality (22) $\mathcal{C}(p_x)$ has to be smaller than $\mathcal{C}(\mathcal{P}_{new})$. Consequently

$$\mathcal{C}(\mathcal{P}_x) < \mathcal{C}(\mathcal{P}_{new}) \leq \mathcal{C}(\mathcal{P}) \Rightarrow \mathcal{C}(\mathcal{P}_x) < \mathcal{C}(\mathcal{P}) \tag{22}$$

Since $\mathcal{P}_x \in P_{calc}$ and $\mathcal{C}(\mathcal{P}_x) < \mathcal{C}(\mathcal{P})$ $\mathcal{P}$ can not have been optimal at t_{calc}, thus having a contradiction.

□

It is now shown that updated size information does not effect optimality.

Lemma 3. *If the size of a previously known object $M_i \in M$ is updated but remains intersection free with the current optimal plan $\mathcal{P}$ the optimality of $\mathcal{P}$ is not affected.*

Proof. Obstacles can only increase in size if new information is detected due to the free space assumption stated in chapter 2: $M_i(t_{cur}) \supset M_i(t_{calc})$. Again $\mathcal{P} \notin P_{inv}$ by assumption.

No new plans can be constructed since no new obstacle or free-space is present, yielding $P_{new} \equiv \emptyset$.

Since $\mathcal{P}$ remains $\in P_{cur}$ and no new plans are added does $\mathcal{P} \equiv \mathcal{P}^*_{calc}$ hold.

□

Lemma 2 and 3 are justifying the optimization step presented in chapter 4.3.1.

5.2.2 Limit Navigation Planner Calls

In the following discussion it will be shown that the bounding of the calculations for an obstacle evaluation does not effecting optimality. This will be done by proving that only plans are being omitted that could not possibly lead to an optimal plan. Lemma 4 will justify the limitation of plan evaluations through the use of $\mathcal{C}_{est}$ as an upper limit. Lemma 5 will in turn show that the limitation to only calculate upon the detection of a new opening is valid.

Lemma 4. *The cost estimate $\mathcal{C}_{est}$ always underestimates the true cost of a plan. $\mathcal{C}_{est}$ is monotonically increasing for additional manipulation actions.*

Proof. $\mathcal{C}_{est}$ is the summation of c_1, $|seq|\mathcal{C}_{man}$ and *euclidianCost*. $|seq|\mathcal{C}_{man}$ is following (15) equivalent to c_2. *euclidianCost* is an underestimate of the true cost. $\mathcal{C}_{est}$ is therefore the summation of actual costs and an underestimate, consequently less or equal the true cost.

Let $\mathcal{C}_{est\_add}$ denote $\mathcal{C}_{est}$ with an additional manipulation action. $\mathcal{C}_{est\_add}$ has equal c_1 to $\mathcal{C}_{est}$. $|seq|$ for $\mathcal{C}_{est\_add}$ increases by one and the euclidean cost can at most be reduced by one navigation action. This yields $\mathcal{C}_{est\_add} - \mathcal{C}_{est} = \mathcal{C}_M$ or $\mathcal{C}_{est\_add} - \mathcal{C}_{est} = \mathcal{C}_M - \mathcal{C}_N$. Following equation (22) $\mathcal{C}_M > 0$ and $\mathcal{C}_M - \mathcal{C}_N > 0$, yielding a positive value for the subtraction.

□

This directly justifies the optimization step described in chapter 4.3.2 (b).

Lemma 5. *A plan $\mathcal{P}_x \in P$ with a non-zero sequence of manipulation actions that does not create a new opening (according to definition 2) in the map can not be optimal.*

Proof. If $P \equiv \emptyset$ this is trivially true, since this indicates that q_r^g is blocked and not reachable. Consequently if no new opening is created by $\mathcal{P}_x$, q_r^g is still not reachable through $\mathcal{P}_x$, which yields $\mathcal{P}_x \notin P$. An evaluation of $\mathcal{P}_x$ is therefore not necessary.

Assume $P \neq \emptyset$. If all all plans in P have a non-zero sequence of manipulation actions than this indicates that the goal configuration q_r^g is only reachable through the creation of a new opening. Further, assume $\mathcal{P}_x$ has a non-zero manipulation sequence on an obstacle while not creating a new opening. Following the domain restriction stated in chapter 3.4.2 can every plan at most have one consecutive sequence of

manipulation actions. $\mathcal{P}_x$ can therefore not successfully transfer the robot to q_r^t. This yields that $\mathcal{P}_x \notin P$ and no evaluation is necessary.

Now, let the set P have at least one plan with no manipulation actions.

Due to the domain restriction in chapter 3.4.2, $\mathcal{P}_x$, has to share a subsequence of navigation actions with a plan only consisting of navigation actions in order to be element of P.

Let $Q \subseteq P$ be the set of valid plans only consisting of navigation actions. Starting from some point after manipulating the obstacle, $\mathcal{P}_x$ has to share an equal subsequence with at least one element in Q, leading to q_r^g. Let $R \subseteq Q \subseteq P$ denote the set of valid plans that have the same subsequence. Let pt denote the point after which the subsequence is equal with $\mathcal{P}_x$.

R has to have at least one element which performs navigation actions to the same grid field where $\mathcal{P}_x$ performs the first manipulation action, denoted as *start_manipulation*, and circulates the obstacle to the point pt. Let r denote such an element. Due to assumption 3.4.4 (goal is always free - obstacle can not be pushed over the goal) $\mathcal{P}_x$ must have navigation actions after the manipulation actions to either

- move to the side of the obstacle,
- in the reverse manipulation direction
- or combination of this.

This is visualized in Fig. 6. Consequently r has to have equal or less distance from *start_manipulation* to pt.

Due to inequality (22) this yields $\mathcal{C}(r) < \mathcal{C}(p_x) \Rightarrow \mathcal{P}_x \neq \mathcal{P}^*$.

□

It is now shown that new openings can only be created adjacent to the obstacle so that the detection becomes independent of the actual map size.

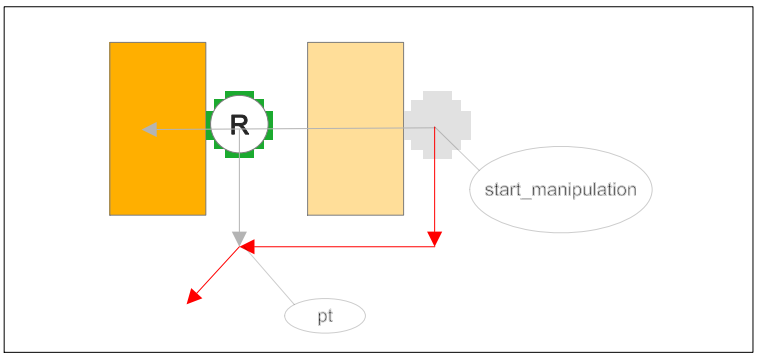


Figure 6: Lemma 6 visualization.

Lemma 6. *Any manipulation action on an obstacle M_i can only create openings directly adjacent to M_i.*

Proof. An opening at time t, denoted as $o(t)$ is a subset of the free-space $o(t) \subset F(t)$. Further, in order to be a new opening must $o(t) \not\subset F(t_{prev})$ at time t_{prev} before the manipulation. Following the definition of free-space (chapter 2) the free-space can only be altered through the manipulation of an M_i. Consequently $o(t) \subset F(t) \wedge o(t) \not\subset F(t_{prev}) \Rightarrow o(t) \cap M_i(t_{prev}) \neq \emptyset \wedge o(t) \cap M_i(t) \equiv \emptyset$. New openings can therefore only appear in space previously occupied by M_i, which now has to be adjacent to M_i. □

Due to assumption 3.4.3 (no obstacle interactions) an obstacle can only be moved directly by the robot, consequently the opening can only occur adjacent to the manipulated obstacle.

Lemma 5 and 6 justify part (a) of the optimization step described in chapter 4.3.2.

5.2.3 Reduce candiate objects

In the following it will be shown that the *minCost* list, explained in chapter 4.3.3, consists of lower bounds as long as no manipulation has been performed since the calculations of the entries.

Lemma 7. *The elements in the minCost are lower bounds for the actual cost for any $\mathcal{P}$ having a non-zero manipulation sequence on M_i, if no manipulation actions have been performed since the last time the elements where modified in the list.*

Proof. Following the definition of *minCost* any $\mathcal{P}$ having a non-zero sequence of manipulation actions on M_i has greater or equal partial cost than the value in the *minCost* list. Further, due to inequality (22) has every $\mathcal{P}$ a non-negative c_1.

If no deviation from assumption 2 is observed at any time $t > t_{calc}$ only c_1 for any $\mathcal{P} \in P_{M_i}$ will change, however it will always remain non-negative and therefore the lower bound holds.

In case of a deviation from the free space assumption the newly as blocked cells detected grid cells will have to be avoided. This is because of assumption 3.4.2 (single obstacle manipulation) and 3.4.3 (no obstacle interactions) no more obstacles can be manipulated. This yields that every plan in P_{M_i} occupying those grid fields becomes invalid. Further, due to assumption 3.4.2 can no new plans be constructed. In consequence will any $\mathcal{P} \in P_{M_i}$ either change just in c_1 or be removed from the set, however no new plans can be added.

Please note that if a manipulation action has been performed after the elements where added to the list, free-space was altered, causing that new plans could potentially be constructed. The elements in *minCost* are not guaranteed to be underestimates anymore. □

Lemma 7 justifies the use of the list *minCost* if no manipulation actions have been performed since the cost values in *minCost* are lower bounds. Upon execution of a manipulation action the entire list is invalidated and therefore each element in the list fulfills the requirements of lemma 7 at each time t.

The euclidean distance will always be an underestimation of the actual cost since free space is assumed everywhere. Therefore the list *euclidianCost* does not have to

be invalidated. This allows the usage of the list *euclidianCost* to efficiently recover a valid *minCost* list without having to evaluate each obstacle. This justifies the optimization step 4.3.3.

It has been shown that none of the optimization steps are affecting the optimality criterion defined in 5. The *optimized algorithm* is therefore optimal.

CHAPTER VI

OPENING DETECTION

The optimization step described in chapter 4.3.2 (a) requires that openings can be detected for the evaluation of manipulation sequences on an obstacle M_i. This chapter will first provide a general discussion about the opening detection process before explaining the opening detection algorithm developed for this work.

6.1 Discussion

Definition 2 was given in accordance to the set $P_{\text{-}}(t)$ and as such in relation to the goal. Every opening detection algorithm used in combinations with the techniques discussed in chapter 4.3.2 has to detect openings according to definition 2.

However, if an opening detection algorithm is returning false positives then this will only lead to additional computations by the algorithm proposed in this work. Optimality will not be affected by false positives. If an opening detection algorithm causes false negatives, on the other hand, optimality could be affected.

Due to the assumptions made in chapter 3.4 the map can only be altered through the robot, and obstacle interactions are restricted. Consequently openings can only occur in relation to the currently manipulated obstacle. An algorithm that is observing the space adjacent to the currently manipulated obstacle is therefore sufficient in finding the true positives while introducing the risk of false positives which is not affecting optimality.

The algorithm developed for this work follows this approach.

6.2 Algorithm

In the following the algorithm developed in this work for the opening detection will be discussed.

6.2.1 Approach

The core idea of the proposed opening detection algorithm is to track areas that prevent the robot from passing the currently manipulated obstacle over different manipulations.

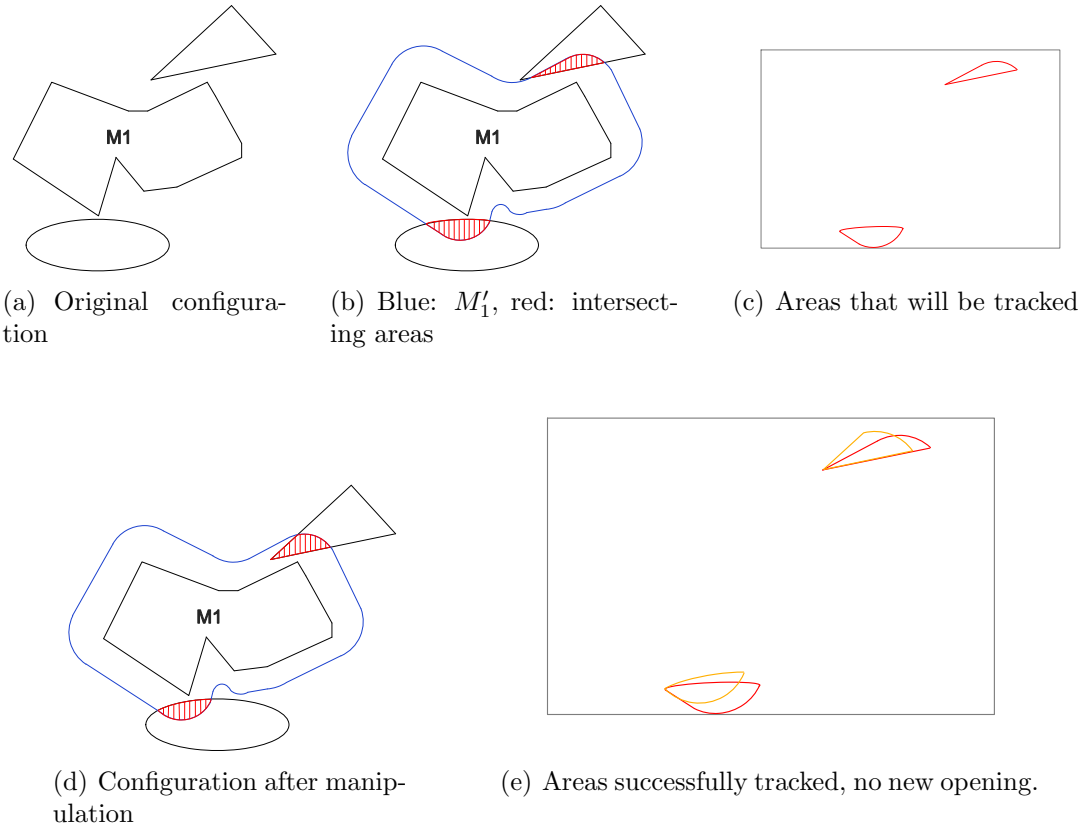
This is done by extending the objects boundaries of the currently manipulated obstacle M_i by the robots dimensions. Let this new representation of the object M_i be denoted as M_i'. Intersections of M_i' with any other obstacles known to the robots are than determined. If such an intersection exists, the intersecting area is tracked over different manipulations. If such an area disappears, the detection of a new opening is declared. As discussed above may this cause false positives according to definition 2, however it will ensure that the manipulation actions causing a new opening as defined by definition 2 will be found. Fig. 15 shows this idea.

6.2.2 Implementation

The following algorithm allows for opening detection on arbitrary shaped obstacles and manipulation directions based on the discussions above. The example shown in Fig. 8 will be used to demonstrate each step of the implementation.

To represent the intersections for the extended obstacle M_i', the algorithm draws the bounding box around M_i'. The cells contained within the bounding box are represented by a matrix F.

If a grid cell is occupied by an object and M_i' and an object M_j, $i \neq j$, the corresponding entry in F is assigned a number unequal zero, otherwise zero. If a number unequal zero has to be assigned, the number is assigned based upon a 3×3 neighborhood of the currently to set entry in F. If a number has already been assigned



(a) Original configuration

(b) Blue: M_1', red: intersecting areas

(c) Areas that will be tracked

(d) Configuration after manipulation

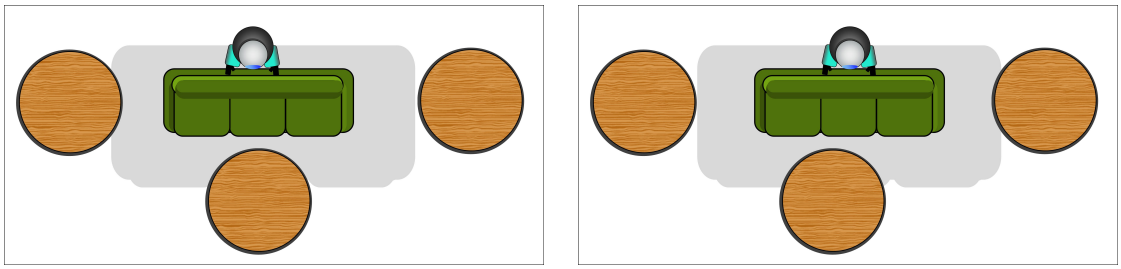
(e) Areas successfully tracked, no new opening.

Figure 7: Example of opening detection

within this neighborhood, the same number is assigned. In case no number has been assigned yet, a number not used in F so far is assigned. For the example in Fig. 8(a) the following matrix is obtained:

$$F = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{2} & \mathbf{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{2} & \mathbf{2} & \mathbf{2} & \mathbf{2} & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The same procedure is performed for the world configuration after the simulated obstacle manipulation. Let E denote the matrix obtained based on the new world configuration. For the example in Fig. 8(b) the following matrix is obtained:



(a) Original configuration, leading to matrix F (b) Configuration after manipulation leading to matrix E

Figure 8: Example setup. The robot is moving the couch to the right and checks for new openings. Gray: the extended object.

$$E = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{2} & \mathbf{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{2} \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The detection of openings is now performed through the following steps:

- the entries in the matrix E are shifted back according to the negative manipulation direction, producing E'. For the example this yields:

$$E' = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

- F and E' are now compared:

– if an entry F_{xy} and E'_{xy} is non-zero for both matrices, all entries in F having the same number as F_{xy} are set to 0. Let F_r denote the resulting matrix. For the example in Fig. 8 the following F_r is obtained:

$$F_r = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

- if F_r equals the zero-matrix after this operation no new opening were detected. If F does not equal the zero matrix one intersecting area could not be found anymore and the possibility of an opening is returned. Since the matrix F_r obtained for example Fig. 8 is not equal to the zero-matrix is an opening detected.

In order to save computation time is M'_i saved for future use. However, if new information about M_i becomes available that changes the shape or dimensions of M_i is M'_i invalidated and recomputed if M_i is evaluated again.

CHAPTER VII

NAVIGATION PLANNING

In this chapter D^* Lite will first briefly be explained. Than, the typical the navigation planning performed by the proposed algorithms will be analyzed. They will be grouped into three different groups. It will than be explained how D^* Lite can be used to enhance the navigation planning of two out of these three groups. This chapter will be concluded with a discussion on the heuristics used for the planners in this work.

7.1 D Lite*

D^* Lite [9] is an incremental heuristic search method for incompletely known environments. It builds upon the *Lifelong Planning* A^* [11] algorithm.

D^* Lite plans from the goal vertex to the start vertex. It maintains two different estimates on this distance. The first, denotes as g, being the estimate of the goal distance, similar to the g value used in the A^* algorithm just in reference to the goal vertex. The second denoted as rhs value (right-hand side value [20]). The rhs value is based on the vertexes predecessors. It is the minimum of the the g values of any of its predecessors and the the edge cost between this predecessor and the vertex. A vertex is called locally consistent if its g value is equal to its rhs value. If all vertices are locally consistent the g values equal the actual goal distance and one can determine a shortest path. However, not all nodes have to be locally consistent. Rather is a heuristic used to guide the nodes that have to be made locally consistent. The heuristic used for D^* Lite has to be nonnegative and backward consistent. A priority queue is maintained with the locally inconsistent vertices. The queue is sorted according to a key with two values. The first value is the smaller of its g and rhs value

plus the heuristic value. The second is just the minimum of the g and rhs value. The keys are used in lexicographical order. D^* Lite expands nodes in the priority queue until the start vertex is locally consistent. Expanding a node sets the nodes g value equal to the rhs value if the g value is larger than the rhs value and as such making the node locally consistent or sets the g value to be infinity. The node will than be made locally consistent upon the next expansion. If edge costs change, it updates the affected vertices rhs value and their position the priority queue and a new path is calculated. Similar, if the robot moves are the entries in the priority queue updated, however this step can be saved with small modifications on the algorithm. See [9] for more details.

7.2 Typical Navigation Planner calls

The NAMO in Unknown Environments domain as described in this work has three typical calls to a navigation planner:

1. from the robots current configuration to the goal configuration,
2. from the robots current configuration to an object manipulation configuration,
3. from the robots configuration after the manipulation of an object to the goal configuration.

The first case is the case most typical for robot navigation. The goal configuration is fixed and the robot has to perform navigation actions from its current configuration until the goal configuration is reached.

The second case is more unique to the NAMO domain and manipulation planning. The navigation goal is not the global goal but rather a goal that is decided online by the robot.

The third case is similar to the first case in the sense that the navigation goal is the global goal, however the start configuration is not equal to the robots current

position.

7.3 Taking use of D^* Lite

Until now A^* was assumed as the navigation planner. However, A^* has the property that each call to it is independent of the previous once. Given the frequency navigation plans are constructed in this work this property can result in a bottleneck.

7.3.1 Navigation to goal configuration

The first case described in the previous section is very suited for D* Lite, since the goal configuration is not changing. However, in NAMO the robot does not always traverse this path, e.g. it navigates to a manipulation configuration. D* Lite has therefor been modified for this domain. The search tree updating has been implemented to be independent of the path calculation. As such D* Lite receives environment updates and the priority queue is updated accordingly, however no new path is computed until explicitly requested by the algorithms described in Chapter 4.

7.3.2 Navigation to manipulation configuration

The case of navigating to a manipulation configuration, if treated in a nutshell, is very similar to the first case in that the robot needs a path to a configuration, and this path might have to be construced multiple times during the execution of the algorithm. A similar concept as for the first case is therefore applied. The robot maintains a D* Lite search tree to the manipulation configurations for each object. The priority queues are again updated but no new path is computed. However, in contrast to the first case, the desired goal configuration can change. This can occur if more of the obstacle becomes visible to the robot. If this is the case, the root of the D* Lite search tree has to change. This has been shown to be a bottleneck of D^* Lite [9]. To avoid this bottleneck, the entire D* Lite searchtree is invalidated and lazily recovered when an actual path is needed. It would also be possible to use Moving

Target D* Lite [27] here; however it seems the additional overhead is not justifiable here given that the manipulation configuration typically just changes when the robot is close to the object and as such a typically small search tree.

The computational overhead for maintaining the D^* Lites for each possible manipulation configuration for each object is minimal. This is because upon environment updated each D^* simply receives the environment updates and just changes the affected nodes *rhs* values and updates the nodes position in the priority queue. However the actual computation of a path and as such iteration over the updated queue is delayed until an actual path is needed. Nevertheless, if desired the *minCost* and *euclidianCost* lists can be used to limit the D^* Lites. For example only the first couple objects that are referenced in this list could have a D^* Lite search tree while the remaining do not have a maintained D^* Lite search tree.

7.3.3 Navigation to goal configuration after obstacle manipulation

The case of planning a navigation path to the goal configuration after planning the manipulation of an object is different than the previously two cases.

In contrast to the other two cases is it necessary to plan the navigation actions in the process of evaluating manipulation actions. It therefore occurs during the internal simulated manipulation of an obstacle. Consequently the navigation planner has to work on a grid world that is not equivalent to the grid representation of the actual world but differs according to the manipulation. Updates to a D^* Lite search tree are therefore updates that are not reflected by actual world updated. Once the obstacle evaluation has terminated, all the updates to the search tree would therefore have to be reverted. In addition would each manipulation action sequence that is evaluated require its own search tree, which would result in an immense amount of copying operations. These facts make it difficult to apply D^* Lite for this case.

Besides the limitations on the usability of D^* Lite for this case it does not usually

present a bottleneck during the actual execution of the algorithm. This is because the navigation planning done for this case typically occurs through the most free space. The free space assumption is combination with the fact that this case only occurs when an opening around the obstacle has been detected lead to this behavior.

Given the difficulty of using D^* or any other incremental search algorithm and the circumstances for the third case, it seems justifiable to rely on A* for the third case.

7.4 Heuristic

Now the heuristic is explained. In the previous chapters the heuristic was stated to be the euclidian distance to the goal configuration. This is an admissable heuristic and can directly be used in all of the algorithms described above mostly independent of the world representation.

The heuristic can however be modeled to take the actual world representation into account. In this work the world was represented as an 8-connected grid, further was assumed that a diagonal navigation action has $\sqrt{2}$ times the cost of an axis aligned navigation action, again denoted as $\mathcal{C}_N$. The above algorithms were therefore implemented using a heuristic that takes this representation into account.

The heuristic is calculated based on the amount of steps a robot has to take in the grid to reach the goal. The following formula was used for a node s and the goal g given the fact that one diagonal navigation action saves two straight navigation actions:

$$h(s,g) = \sqrt{2}C_N dS + C_N(m - 2dS) \tag{23}$$

with

$$dS = min(|s_x - g_x|, |s_y - g_y|) \tag{24}$$

$$m = |s_x - g_x| + |s_y - g_y| \tag{25}$$

dS therefore represents the possible diagonal navigation actions and m the manhattan distance.

If assigning a different cost for diagonal navigation actions inequality (22) has to be changed accordingly. May $\mathcal{C}_N$ be the cost of an axis aligned navigation action and $\mathcal{C}_{Nd}$ the cost for a diagonal navigation action than:

$$\mathcal{C}_M > max(\mathcal{C}_N, \mathcal{C}_{Nd}) \geq 0 \tag{26}$$

has to hold.

CHAPTER VIII

EXPERIMENTS

In order to validate the effectiveness of the proposed algorithm and each optimization step, a series of experiments were conducted.

This chapter will first evaluate the difference between the baseline and the optimized algorithm before analyzing each of the optimization steps independently. All the experiments where performed on a machine equipped with an Intel Core 2 Duo (2.93GHz) processor and 2GBs of RAM. If runtime is discussed, the values refer to the time between the moment the simulation was started until the robot reaches the goal. This includes actual movements of the robot. Obstacle evaluations refers to the amount of times the search for a p_{optM_i} was triggered.

8.1 Algorithm comparison

The algorithms were compared on maps of different sizes and complexity However, basically searching the entire search space as the baseline algorithm does, becomes unfeasible for even small maps, for example the map shown in Fig. 9. The robot first tries to circumvent the vertical couch before detecting the second horizontal couch. The robot online decides to move the horizontal couch to the right in order to be able to walk between the couches. Using the *baseline* algorithm it takes the robot more than 2 hours to reach the goal in this map. During the execution the navigation planner is called over 72,000 times and the vertical and horizontal couches was evaluated 46 and 43 times respectively. The optimized algorithm reduces the time it takes the robot to reach the goal (again, including the actual movement) to just 18 seconds while only calling the navigation planner 102 times. The vertical couch was evaluated only 4 times while the horizontal couch was evaluated 32 times.

Please notice that multiple calls are necessary since the robot detects its environment step by step. Further, the goals and objects are very close together, which does not allow for a very clear decision if the object should be manipulated and the robot has to re-evaluate quite frequently since possible manipulation actions cannot be rejected before hand based on estimates.

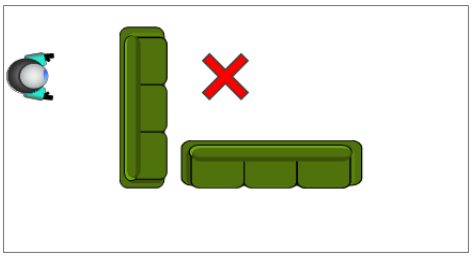


Figure 9: Realistic setup with two couches

However, as the map size increases and the number of obstacles increases, the differences become even more distinct. This is because the explored portion of the search space does not necessarily increase for the optimized algorithm for a bigger map due to the fact that many obstacles and actions are never evaluated. This is not true for the baseline algorithm. The entire search space has to be searched, and as such bigger maps necessarily increase the search space. Fig. 1 shows an example of a bigger map. The baseline algorithm was run on a map very similar to Fig. 1. The optimized algorithm caused the robot to reach the goal within 50.2 seconds and only 112 navigation planner calls. The baseline algorithm took for the same map 5 weeks, 2 days and 6 hours with more than 1,413,700 navigation planner calls. The optimized algorithm triggered 66 obstacle evaluations while the baseline algorithm triggered 2348. The average savings for 5 maps of different sizes, ranging from maps similar to Fig. 9 to Fig. 1, can be seen in table 1.

The baseline algorithm proves to not be practically for realistic scenarios.

Table 1: Average savings for the optimized vs baseline algorithm

Average savings	
Runtime	99.84%
Obstacle Evaluations	88.70%
Navigation Planner Calls	99.98%

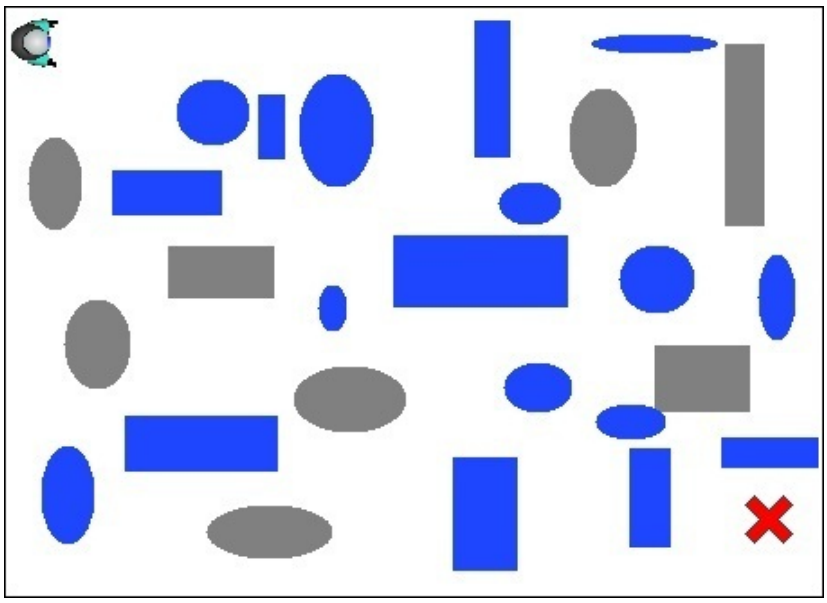


Figure 10: Complex setup. Blue obstacles are movable while gray obstacles are static

8.2 Optimization Steps

Each optimization step is now evaluated separately. However, this was done by *turning one optimization step off* while still relying on the remaining optimization steps since the runtime would have been to high otherwise. The following results therefore have to be interpreted as 'how much and when the optimization step adds additional savings' to the algorithm. Evaluation for each step was performed on 50 randomly generated maps ranging in complexity from configurations similar to Fig. 9 up to maps with more than 70 obstacles. The graphs are interpolated and bezier smoothed to allow a clearer observation of the tendencies.

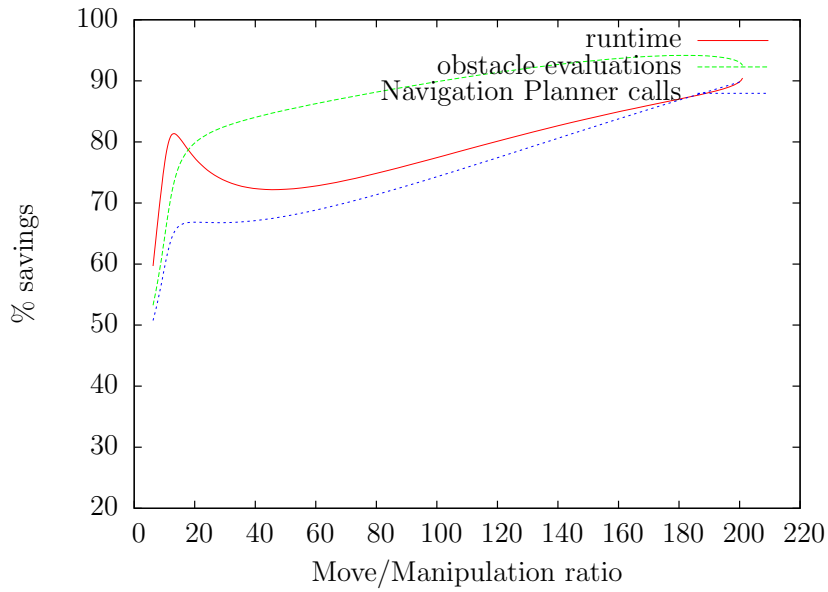


Figure 11: savings by using *minCost* and *euclidianCost* compared to the optimized algorithm without the lists

8.2.1 Reduce candidate objects

The optimization step using the *minCost* and *euclidianCost* lists heavily depends on the map configuration. For maps with sparse obstacle placements almost all the obstacle evaluations can be eliminated, however if many obstacles are close together all with similar estimated cost it might not be possible to eliminate them as candidates that could yield a plan with smaller cost. This was tried to be captured in the ratio of move actions to manipulation actions the robot has to perform to reach the goal. Due to the observation that a path mainly consisting of move actions mostly appear on maps where obstacles are sparsely placed or easily circumventable. Fig. 11 shows the results. A clear tendency can be observed that for maps with a high move to manipulation action ratio the savings increase as expected.

Table 2 shows the average savings over all maps compared to the optimized algorithm without the lists.

Table 2: Average savings for the use of *minCost* and *euclidianCost*

Average savings	
Runtime	78.08%
Obstacle Evaluations	81.68%
A^* Calls	70.65%

8.2.2 Limited Navigation Planner calls

In the following the modifications to the obstacle evaluation procedure are evaluated. First the usefulness of the upper bound is evaluated and later the opening detection.

8.2.2.1 Upper bound

The savings compared to the baseline algorithm mainly depend on the map size. Fig. 12 therefore shows the averaged additional savings dependent on the map size, opening detection was still used for this graph.

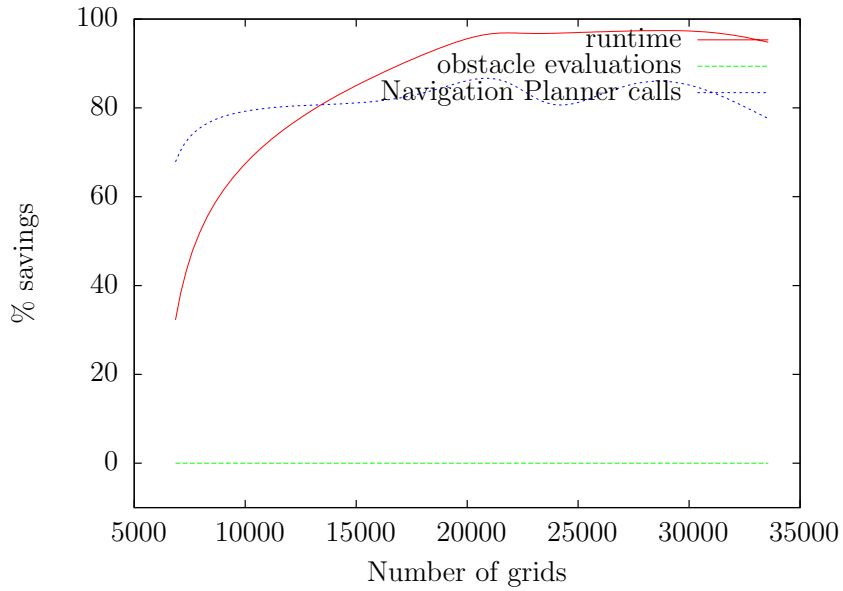


Figure 12: Additional savings if the upper bound is used in the optimized algorithm compared to the baseline algorithm.

It can be observed that the upper bound has a significant impact for bigger maps. Please notice that this optimization step does not alter the number of obstacle evaluation calls. In addition, it can be observed that the runtime savings increase besides

the fact that the navigation planner call savings remain almost constant. This is mainly caused by the optimization step that only navigation plans are constructed if openings in the map occur, as described above. A larger map size does therefore not necessarily indicate more navigation planner calls, but rather is this quantity mainly influenced by the actual map setup. However, if the upper bound is not used, all possible manipulations have to be constructed, yielding a very high runtime difference.

8.2.2.2 Opening detection

Fig. 13 shows again the savings vs the ratio of move and manipulation actions. It can be observed that for maps with a high move to manipulation ratio the savings decrease. This is because of the affect discussed in the optimization step of reduced candidate objects. As can be seen in Fig. 11 the amount of evaluated obstacle almost goes to zero (savings approach 100%) and as such most of the navigation planner calls are not caused by obstacle evaluations anymore. They are rather just caused by the initialization of upper bound, constructing a pure navigation plan, and the lists than prevent an actual obstacle evaluation and the effect of this optimization step.

Further, the runtime savings are not quite as high as the savings for the navigation planner calls, this is because of the computational overhead of detecting the openings.

This optimization step does not affect the number of obstacle evaluations. The average savings are again summarized in table 3.

Table 3: Average savings for opening detection

Average savings	
Runtime	53.59%
Obstacle Evaluations	0%
A^* Calls	54.24%

8.2.3 Recalculation triggering

A similar graph as used for the evaluation of the previous two optimization steps can be seen in Fig. 14 and the average results are listed in table 4. It is especially

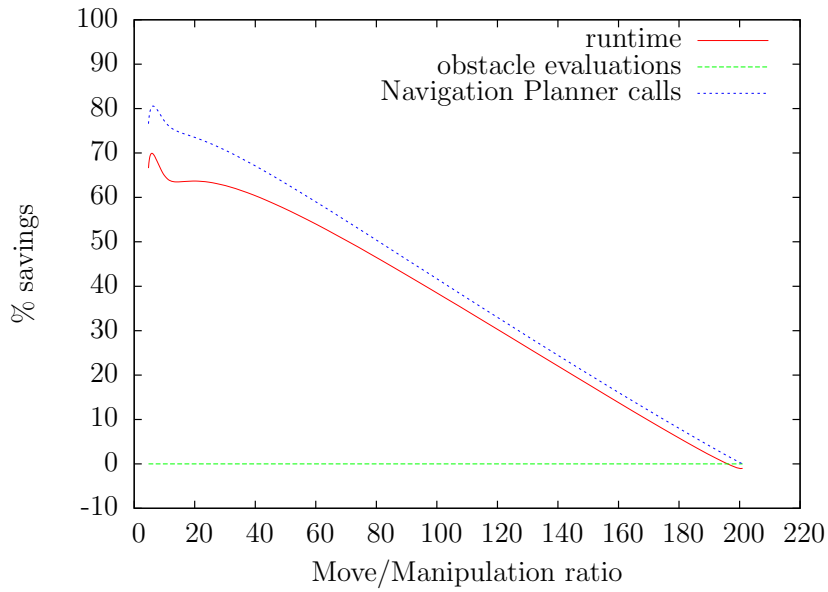


Figure 13: savings if opening detection is used

interesting to see that the average savings are the lowest for this optimization step in comparison to the other optimization steps. This is mainly caused by the fact that even if calculation is triggered upon any update of environment information, the other optimization steps, especially the usage of *minCost* ensures that the necessary calculations are very limited.

The runtime drop in Fig. 14 is mainly caused by simulator specific calculations that occur for bigger maps (e.g rendering, grid calculations etc). Bigger maps that had to be used to increase the ratio. The drop is not visible in the previous graphs because most of the computation was caused by the planning. Here, in contrast, planning is almost entirely eliminated by the remaining optimization steps causing the simulator specific calculations to become dominate.

Table 4: Average savings if calculation is only triggered once the current plan is intersected.

Average savings	
Runtime	43.62%
Obstacle Evaluations	53.38%
A^* Calls	63.18%

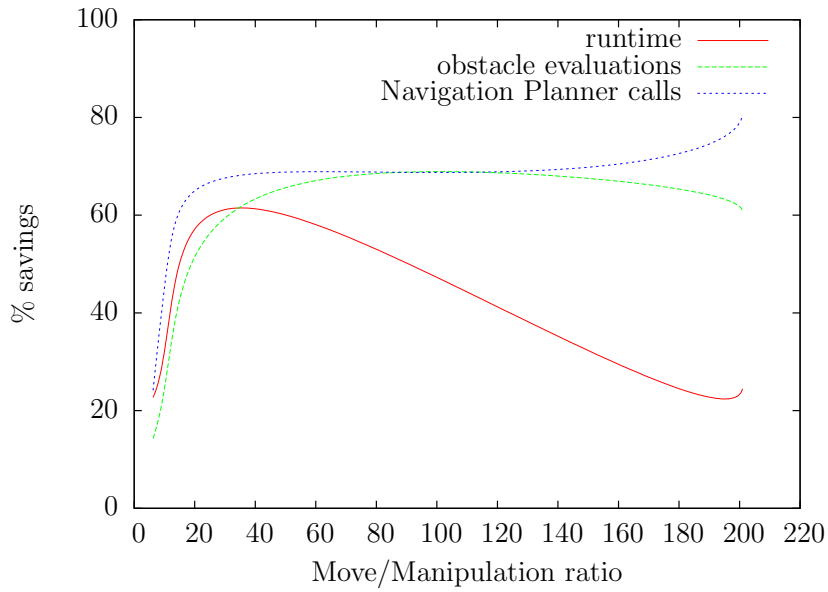


Figure 14: savings if the calculation of a plan is only triggered if the the currently optimal plan is intersected

8.3 Examples

Fig. 15(a) shows an example setup with more than 30 obstacles. The goal is not reachable without manipulation. At the beginning of the execution the robot only knowns the position of the goal but has no information about the objects. Fig 15(b) shows the robot during the execution. Colored objects represents objects or partial objects known to the robot. The obstacles in light blue and gray are not known to the robot yet. The robots sensor range is visualized through a red circle around the robot. The robots plan it has been chosen for execution is visualized. Navigation actions are shown in red. One can see that the robot plans navigation actions through obstacles that he has not yet encountered due to the free space assumption. Further has the robot in Fig. 15(b) already moved the couch right above it since it has detected that it can not reach the goal otherwise.

Fig. 15(c) shows the robots executed actions mapped into the original map configuration. Again, navigation actions are shown in red, additionally are manipulation actions shown in orange. One can see that the robot has manipulated three objects

before reaching the goal.

During the execution of this map using the optimized algorithm the navigation planner was called 203 times and obstacles were evaluated 75 times for possible manipulations. The robot reached the goal within 63 seconds. The baseline algorithm was not capable of solving this map besides a runtime of multiple days.

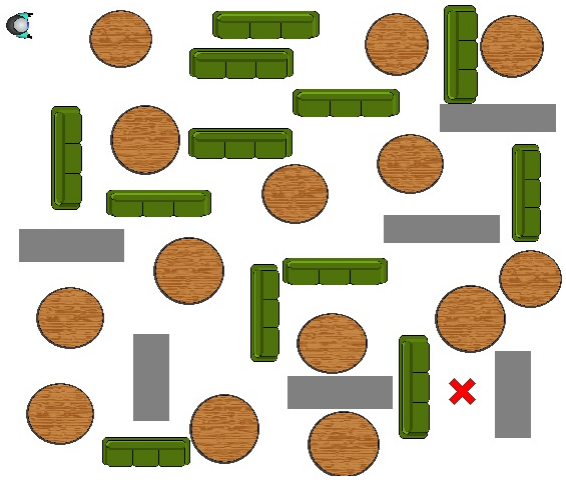
8.3.1 Multiple obstacle evaluations

In addition to the previous discussed reasons for multiple obstacle evaluations (obstacles close to the goal, yielding almost a tie in cost values) can the repeated calls be caused by problem inherit to unknown environments while requiring optimality. If the robot plans to manipulate an obstacle it has to navigate towards the manipulation configuration. Further, as discussed above does the robot plan to manipulate an object in order to create a new opening in the map. An optimal plan usually yields a navigation actions that passes right next to the obstacle after the opening has been created. Now, if the robot actually has not encountered the full object yet, it will detect more parts of the obstacle. In this situation the newly detected object information will in most cases intersect with the current plan, triggering recalculation. These recalculations in turn are causing obstacle evaluations to be executed on the obstacle on which a manipulation was planed in the previous plan because it is very likely that it could still yield a lower cost plan to the goal. Fig. 16 shows a typical example of multiple obstacle evaluations.

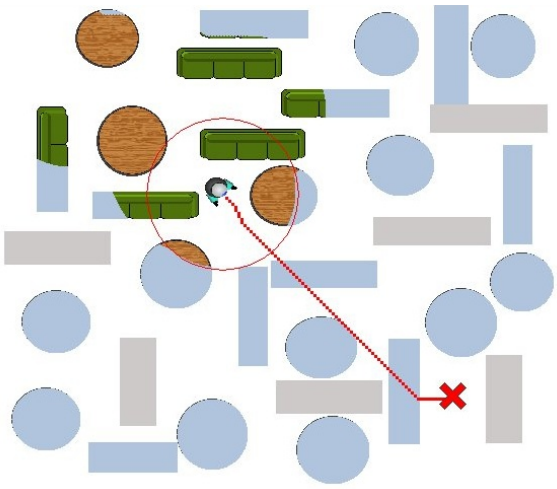
A similar, if the robot detects more of the obstacle it is also very likely that the manipulation action is changing based on the requirement that the obstacle hast to be grasped at the center of one of the axis aligned sides of the obstacle. This again will almost certainly trigger obstacle evaluations on at least the object that was previously planned to be manipulated.

These cases are difficult to avoid. One could stall re-planning while the robot

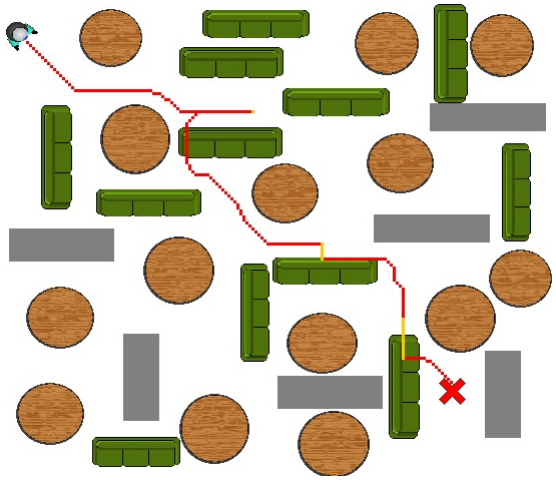
is navigating towards a manipulation configuration if newly detected information is not interfering with navigation plan towards the navigation configuration. However, optimality could than obviously not be guaranteed anymore.



(a) Original map configuration.



(b) The robots knowledge of the world. Colored objects are known while grayed objects are unknown. Red circle around the robot visualizes its sensor range.



(c) Executed steps mapped into the original setup.

Figure 15: Example with more than 30 objects.

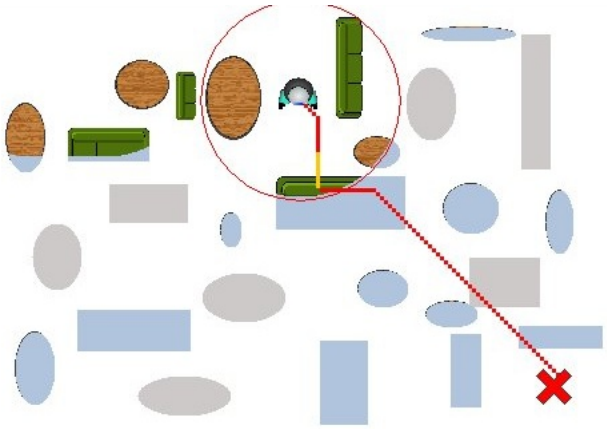


Figure 16: Multiple obstacle evaluation caused by incomplete knowledge of the object that is to manipulate.

CHAPTER IX

CHALLENGES

This chapter will detail on challenges inherent to the the domain of Navigation Among Movable Obstacles in Unknown Environments as well as challenges introduces by the restrictions made in this work.

9.1 Inherent Challenges

The domain of NAMO with incomplete information has unsolvable problems. If the robot is given only partial information, the robot cannot avoid all the negative effects resulted from reconfigurations. The robot makes its decisions based on the assumption that unknown space is free space. This assumption can cause the robot to reconfigure its environment in such a way that it actually hardens the global problem. Local solutions will not always solve the global problem.

Secondly, the presented planner is limited to at most manipulating one obstacle for every plan. If this limitation would not be given the robot would have the capability of potentially reversing previous manipulations that potentially block necessary further manipulations after detecting more objects.

9.2 Example

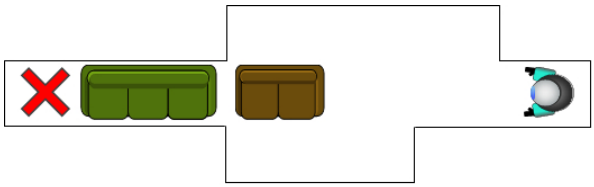
The scenario in Fig. 17(a) demonstrates an example where local information prevents the robot from finding a global solution. Since the robot only has partial knowledge about the world it will first just detect the brown couch as a blocking obstacle (Fig. 17(b)). Based upon this information the couch can be moved up. Unfortunately, if the couch is moved up the goal can not be reached anymore, as seen in Fig. 17(d). If the the robot would have moved the brown couch down it would have avoided

to block itself and the goal could be reached, see Fig. 17(c). This fact can not be discovered by the robot due to the partial knowledge of its environment. This is an inherit problem introduced by the partial knowledge.

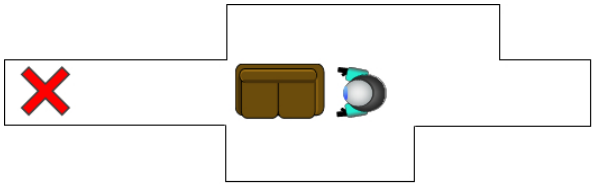
In same scenario also demonstrates the limitations introduces by the restriction a plan to manipulate at most one object. The presented planner is limited to at most manipulating one obstacle for every plan. If this limitation would not be given the robot could, after moving the brown couch up and detecting the green couch move the brown couch down and than move the green couch back and as such find the solution in Fig. 17(d) for a plan involving two obstacle manipulations.

In addition, ff the robot would detect both objects at the same time the proposed planner would fail since no single object manipulation can yield a path to the goal given the current world knowledge. This example shows two restrictions of the domain.

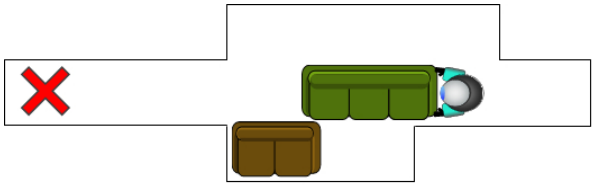
In summary, the presented approach faces two major challenges. First, given the premise that partial information is complete, the best solution for the currently known environment does not necessarily solve the global problem. Reconfigurations can even block the solution. Second, the limitations to single object manipulations can result in unsolvable cases.



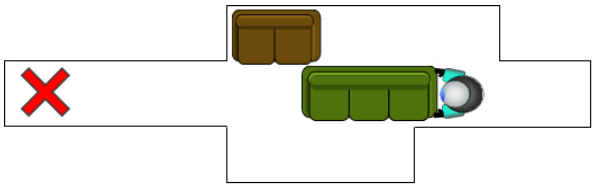
(a) Map configuration



(b) Partial knowledge



(c) If the brown couch was moved down, the goal can be reached.



(d) If the brown couch was moved up, the goal is blocked

Figure 17: Example of the limitations of NAMO in Unknown Environments.

CHAPTER X

FUTURE WORK

This chapter will give an outline of how the domain of Navigation Among Movable Obstacles in Unknown Environments could be extended to allow multiple object manipulations within a plan. Further will practical aspects be addressed that have to be handled to transfer the proposed planer to a physical robot.

10.1 Multiple Objects

The algorithms introduced in this work are capable of handling arbitrary displacements and obstacles of arbitrary shape. However each plan $\mathcal{P}$ is restricted to have at most one consecutive sequence of manipulation actions on one obstacle. While the fact that obstacles are detected over time - yielding that multiple plans might be partially executed - allows the robot to practically manipulate multiple objects before reaching the goal, this restriction is limiting as was shown in chapter 9.

Further work in the domain of NAMO in Unknown Environments should therefore drop this restriction.

The search space reduction techniques presented in this work could potentially be used to extend the domain to multiple object manipulations. One such approach could be to formulate the entire NAMO in Unknown Environments problem has a heuristic search. For example, could the movable objects as well as the goal are represented as the nodes in a search tree. The edges, connecting the nodes, represent the possibility to navigate from one object to the next. The expansion of a node is now done through the techniques described in this work. To expand a node, all the possible manipulation actions on the obstacle are evaluated up to an upper bound and the openings are detected. For each opening the navigation cost to all the other objects

are computed. The cheapest combined cost of manipulating costs and navigation costs to an object represent the edge cost to the node representing the next object. To guide this search a list similar to the here introduced *minCost* lists could be used. Once the goal is chosen for expansion the algorithm terminates.

This approach would require a fast navigation planner, potentially reusing as many information from previous search as possible. Techniques would have to be developed to explicitly take use of the fact that the search space is incrementally increasing over the runtime and very manageable at the beginning of the execution.

10.2 Uncertainty

This work is investigating the domain of Navigation Among Movable Obstacles in Unknown Environments. The robot does not have any initial knowledge about the world and just gains knowledge incrementally as more of the world become accessible by his sensors. However, once information is perceived by the sensors it is taken to be correct. The new information is incorporated in the internal map which is also taken to be correct for all the parts that have been observed yet. The robot therefore assumes his current map to be correct at all times. Any real sensor system however will not be able to always and only return correct information. This should be accounted for. Further is always assumed in this work that the grasping of an obstacle and its manipulation will succeed. Again, in real systems this is not given.

Further work should therefore try to extend the domain of Navigation Among Movable Obstacles in Unknown Environments to the domain of Navigation Among Movable Obstacles in uncertain environments. This domain should account for the uncertainty introduced by a real robot system. Taking the uncertainty of the sensor system into account will lead to different decisions by the robot. The robot can now in addition take its certainty into account. For example may it be better to avoid certain obstacles if the uncertainty about such these obstacles is exceeding a threshold

and a successful manipulations seems unlikely. In addition could the robot decide to perform actions with the premises to gain more information about an obstacle prior to making a decision about manipulating it. The Partially Observable Markov Decision Processes (POMDP) [1] could potentially be modified to be used for this domain.

10.3 Free space

The robot in this work assumes free space for unknown space. This yields two problems. As described in chapter 8.3, multiple obstacle evaluations are caused by the robot detecting more of an object while navigating towards the object in order to manipulate it. The impact of this could be reduced by making assumptions about the remaining parts of the object, even though they have not been within the sensors range yet. The robot could than plan more efficiently.

Secondly the robots knowledge about the observed parts of the world could be used to make assumptions about the unobserved space. The robot can then adopt its cost functions depending on the environment. For example if the robot has observed that it is in an environment that does not seem very cluttered, it might bias its decision of manipulating an object towards circumventing it because it is seems unlikely that a detour would encounter many more objects. On the other hand, if the robot seems to be in a very cluttered environment it may try to shorten its path as much as possible because it seems likely to encounter objects very frequently.

CHAPTER XI

CONCLUSION

This work has introduced the first practical and optimal planner for the domain of Navigation Among Movable Obstacles in Unknown Environments. The planner is capable of working with objects of arbitrary shape and can handle arbitrary displacements.

The *basline* algorithm was introduced. The baseline algorithm is a naïve algorithm that allowed for optimal decision making in the domain. The *baseline* algorithm computes upon any new environment information all possible plans with the exceptions of plans that only differ in the navigation actions. The final plan is chosen to be the plan with the minimum cost as defined through a cost function. It was shown that optimality can be guaranteed for this planner. However, the planner proved to not be practical even for moderately sized environments with a runtime of multiple weeks.

Consequently the *optimized* algorithm was introduced. The optimized algorithm still guarantees optimality while drastically reducing the search space. The *optimized* algorithm introduced three major differences to the baseline algorithm.

First re-planning is only performed if the newly detected environment information is intercepting the current plan.

Secondly the algorithm determines an upper bound on obstacle evaluations. The upper bound is set to be the value of the currently known minimum cost plan and is checked against a final cost estimate of a plan. Further are only such plans considered that are creating an opening in the map.

Finally, two lists are maintained that represent underestimates of plans having a non-zero sequence of manipulation actions on an obstacle referenced in the lists. The

list *minCost* represents tighter bounds than the list *euclidanCost* as the list includes partial costs actually calculated. The *euclidanCost* list only saves a distance estimate to the goal while assuming free-space. Due to the fact that free space can be created in the domain of Navigation Among Movable Obstacles and as such alter elements in the *minCost* list is the list invalidated when manipulation actions are performed. The *euclidianCost* is used to lazily recover the *minCost* list.

This work also introduced concepts for the navigation planning typically occurring in this domain. A combination of different D^* Lite trees was used to ensure that information from previous searches are being reused.

The detection of openings, as necessary for the *optimized* algorithm, was discussed. An algorithm was presented that efficiently performs this opening detection based on matrices representing the areas that prevent the robot from passing by the currently manipulated object.

The *baseline* algorithm was compared against the *optimized* algorithm showing saving in runtime of more than 99.8% for environments were the baseline algorithm was still applicable. Each of the optimization steps introduced in this work were than evaluated independently. It was shown that each of the optimization steps has its strength in different areas. The step reducing the re-planning triggering was shown to be the least effective if the remaining optimization steps are still being used. This is because the remaining optimization steps ensure that re-planning will terminate quickly if the new environment information is not affecting the current plan.

Challenges of the domain itself as well as limitations of the proposed algorithms were discussed. It was shown that given only local information, a goal solution can not always be found. Limitation of the algorithm to only manipulate one object within one plan was shown to yield cases that are not solvable. During the entire execution however the robot may manipulate multiple objects before reaching the goal.

Future work was presented for not just extending this domain but also incorporating the uncertainty introduced by physical systems.

REFERENCES

[1] Cassandra, A. R., Kaelbling, L. P., and Littman, M. L., "Acting optimally in partially observable stochastic domains," 1994.

[2] Chen, P. and Hwang, Y., "Practical path planning among movable obstacles," in *In Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 444–449, 1991.

[3] Demaine, E., O'Rourke, J., and Demaine, M. L., "Pushpush and push-1 are np-hard in 2d," in *In Proceedings of the 12th Canadian Conference on Computational Geometry*, pp. 211–219, 2000.

[4] H.Wu, Levihn, M., and Stilman, M., "Navigation among movable obstacles in unknown environments," in *IEEE/RSJ Int. Conf. On Intelligent Robots and Systems (IROS 10)*, October 2010.

[5] Igarashi, T. and Stilman, M., "Homotopic path planning on manifolds for cabled mobile robots," in *Algorithmic Foundations of Robotics IX* (Hsu, D., Isler, V., Latombe, J.-C., and Lin, M., eds.), vol. 68 of *Springer Tracts in Advanced Robotics*, pp. 1–18, Springer Berlin / Heidelberg, 2011.

[6] in Seoul National University, R. L., "Snu robotics library." `http://r-station.co.kr/forum/`.

[7] Kakiuchi, Y., Ueda, R., Kobayashi, K., Okada, K., and Inaba, M., "Working with movable obstacles using on-line environment perception reconstruction using active sensing and color range sensor," in *Intelligent Robots and*

Systems (IROS), 2010 IEEE/RSJ International Conference on, pp. 1696 –1701, 2010.

[8] Koenig, S., *Goal-Directed Action with Incomplete Information*. PhD thesis, 1997.

[9] Koenig, S. and Likhachev, M., "Improved fast replanning for robot navigation in unknown terrain," in *in Proceedings of the International Conference on Robotics and Automation*, pp. 968–975, 2002.

[10] Koenig, S. and Likhachev, M., "Fast replanning for navigation in unknown terrain," *Robotics, IEEE Transactions on*, vol. 21, no. 3, pp. 354 – 363, 2005.

[11] Koenig, S., Likhachev, M., and Furcy, D., "Lifelong planning a*," *Artif. Intell.*, vol. 155, pp. 93–146, May 2004.

[12] LaValle, S., "Robot motion planning: A game-theoretic foundation," *Algorithmica*, vol. 26, no. 3-4, pp. 430–465, 2000.

[13] Li, Y. and Li, T., "A unified approach to planning versatile motions for an autonomous digital actor," *JACIII*, vol. 12, no. 3, pp. 277–283, 2008.

[14] Lumelski, V. and Stepanov, A., "Dynamic path planning for a mobile automaton with limited information on the environment," *IEEE Transactions on Automatic Control*, vol. AC-31, no. 11, pp. 1057–1063, 1986.

[15] Ng, J. and Bräunl, T., "Performance comparison of bug navigation algorithms," *J. Intell. Robotics Syst.*, vol. 50, no. 1, pp. 73–84, 2007.

[16] Nieuwenhuisen, D., van der Stappen, A., and Overmars, M., "An effective framework for path planning amidst movable obstacles," *Algorithmic Foundation of Robotics VII*, pp. 87–102, 2008.

[17] Okada, K., Haneda, A., Nakai, H., Inaba, M., and Inoue, H., "Environment manipulation planner for humanoid robots using task graph that generates action sequence," in *In: Proceedings of 2004 International Conference on Intelligent Robots and Systems*, pp. 1174–1179, 2004.

[18] Pirjanian, P., "Behavior coordination mechanisms – state-of-the-art," 1999.

[19] Pirjanian, P., "The notion of optimality in behavior-based robotics," 1999.

[20] Ramalingam, G. and Reps, T., "An incremental algorithm for a generalization of the shortest-path problem," in *J. Algorithms*, vol. 21, pp. 267–305, 1996.

[21] Stentz, A., "Optimal and efficient path planning for partially-known environments," in *In Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 3310–3317, 1994.

[22] Stentz, A., "The focussed d* algorithm for real-time replanning," in *In Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 1652–1659, 1995.

[23] Stilman, M. and Kuffner, J., "Navigation among movable obstacles: Real-time reasoning in complex environments," in *Proceedings of the 2004 IEEE International Conference on Humanoid Robotics (Humanoids'04)*, vol. 1, pp. 322 – 341, December 2004.

[24] Stilman, M. and Kuffner, J., "Planning among movable obstacles with artificial constraints," in *WAFR*, pp. 119–135, 2006.

[25] Stilman, M., Nishiwaki, K., Kagami, S., and Kuffner, J., "Planning and executing navigation among movable obstacles," in *IEEE/RSJ Int. Conf. On Intelligent Robots and Systems (IROS 06)*, pp. 820 – 826, October 2006.

[26] Sun, X., Koenig, S., and Yeoh, W., "Generalized adaptive a*," in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 1*, AAMAS '08, (Richland, SC), pp. 469–476, International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[27] Sun, X., Yeoh, W., and Koenig, S., "Moving target d* lite," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, AAMAS '10, (Richland, SC), pp. 67–74, International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[28] Van Den Berg, J., Stilman, M., Kuffner, J., Lin, M., and Manocha, D., "Path planning among movable obstacles: a probabilistically complete approach," *Workshop on Algorithmic Foundation of Robotics (WAFR VIII)*, pp. 599–614, 2008.

[29] Wilfong, G., "Motion planning in the presence of movable obstacles," in *SCG '88: Proceedings of the fourth annual symposium on Computational geometry*, (New York, NY, USA), pp. 279–288, ACM, 1988.